VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2021 Jaromír Szymutko

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

IMPLEMENTACE SLUŽBY PRO ZPRACOVÁNÍ DIAGNOSTICKÝCH DAT Z AUTOMOBILU
IMPLEMENTATION OF A SERVICE FOR PROCESSING DIAGNOSTIC DATA FROM A CAR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Jaromír Szymutko**
AUTHOR

VEDOUCÍ PRÁCE — **Mgr. Ing. Pavel Šeda**
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Telekomunikační a informační systémy**

Ústav telekomunikací

Student: Jaromír Szymutko ***ID:*** 211273
Ročník: 3 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Implementace služby pro zpracování diagnostických dat z automobilu

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je vytvořit službu, která bude zpracovávat diagnostická a informativní data z automobilu a bude schopná poskytovat statistiky nad těmito daty. Mezi tato data bude patřit např. informace o servisním intervalu, aktuální a průměrná spotřeba, pozice tankování a další. Součástí analýzy dat bude i funkce detekce odtahu vozidla a krádeže. Data budou ukládána do navržené databáze. Aplikace bude poskytovat RESTové rozhraní, pomocí kterého budou data poskytována klientským aplikacím. Výsledné řešení se bude skládat ze služby implementované v jazyku Java za použití framework Spring a z webové části služby pro prezentaci a ovládání implementovanou v Javascriptové knihovně React. Na závěr student otestuje vytvořený prototyp nad testovacími daty, které si sám, po konzultaci s vedoucím práce, vygeneruje.

DOPORUČENÁ LITERATURA:

[1] WALLS, Craig. Spring in Action. Fifth Edition. Manning Publications, 2019. ISBN 9781617294945.

[2] KEITH, Mike, Merrick SCHINCARIOL a Massimo NARDONE. Pro JPA 2 in Java EE 8: An In-Depth Guide to Java Persistence APIs. 3rd Edition. Apress, 2019. ISBN 9781617294945.

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 31.5.2021

Vedoucí práce: Mgr. Ing. Pavel Šeda

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

ABSTRAKT

Bakalářská práce se zabývá vývojem webové aplikace, která má za úkol získávat a vyhodnocovat diagnostická a informativní data z automobilu. Serverová část této aplikace byla vyvinuta v programovacím jazyce Java s podporou frameworku Spring. Tato část zajišťuje veškerou aplikační logiku a propojení s databází. Uživatelské rozhraní bylo vytvořeno pomocí HTML šablon a k nim přidružených CSS souborů pro stylizaci. Dynamické prvky byly implementovány pomocí skriptů v programovacím jazyce JavaScript. Aplikace umožňuje uživateli registraci s potvrzením, přihlášení, registraci automobilu, sledování diagnostických a informativních dat přicházejících z automobilu a přidání informací do profilu uživatele. Data z těchto hlášení jsou přehledně prezentována v tabulkách, případně v grafech.

KLÍČOVÁ SLOVA

OBD II, autodiagnostika, Java, Spring, Hibernate, REST API, HTML, CSS, JavaScript

ABSTRACT

This bachelor's thesis is focused on the development of a web application designed for gathering and processing diagnostic and informative data from a vehicle. The backend was developed in Java with the support of the Spring framework. This part of the application contains all the application logic and provides a connection to the database. The frontend was developed as HTML templates with CSS style sheets. Dynamic elements were implemented as JavaScript scripts. The application supports user registration, user login, registration of the vehicle, and adding information to the user's profile. It also presents diagnostic and informative data from the vehicle in a user-friendly way by using tables and plots.

KEYWORDS

OBD II, car diagnostic, Java, Spring, Hibernate, REST API, HTML, CSS, JavaScript

SZYMUTKO, Jaromír. *Implementace služby pro zpracování diagnostických dat z automobilu* . Brno, 2021, 58 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Mgr. Ing. Pavel Šeda

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Implementace služby pro zpracování diagnostických dat z automobilu “ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Mgr. Ing. Pavlovi Šedovi za cenné rady, pohotové reakce na mé podněty a trefné připomínky k práci.
Dále bych také rád poděkoval své rodině a přítelkyni za trpělivost a neutuchající podporu během celého studia.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Od vynálezu automobilu uplynulo již téměř 160 let a dá se říci, že změnil svět. Z počátku se jednalo o velmi jednoduchá vozidla bez jakékoliv elektroniky, snad kromě zapalovací soustavy. Za celou dobu svého vývoje prodělaly automobily rapidní změny, které reflektovaly vynálezy doby, ve které byly konstruovány. Časem přibývalo v automobilech stále více elektroniky. Proto bylo potřeba vytvořit elektronický systém, který by dokázal diagnostikovat závady jak elektronické, tak mechanické. V dnešní době jsou automobily protkány elektronikou, stejně jako svět kolem nás. Proto je nasnadě vytvoření webové aplikace, pomocí níž je uživatel schopen okamžitě zjistit stav svého vozidla.

Cílem této práce je vytvořit webovou aplikaci za pomocí frameworku Spring v programovacím jazyce Java. Tato aplikace bude mít za úkol přijímat diagnostická a informativní data z automobilu pomocí REST API. Tato data následně uloží do databáze a vyhodnotí je. Její uživatel tak bude mít prostřednictvím přehledného uživatelského rozhraní přehled o statistických datech svého automobilu, jako jsou například dlouhodobá průměrná spotřeba, ujetá vzdálenost a servisní interval. Uživatel bude upozorněn službou i v případě nenadálých událostí, jako je například porucha, odtah či krádež vozidla.

1 Úvod do automobilové diagnostiky

„Automobilová diagnostika je cílený postup, který vede k odhalení závady na voze nebo k nastavení či změnám konfigurací jednotlivých zařízení [1].“

1.1 Historie vzniku automobilové diagnostiky

Historie diagnostických zařízení se začala psát již na konci 60. let 20. století, kdy výrobce automobilů Volkswagen přišel s prvním prototypem diagnostického zařízení, které montoval do svých modelů [2]. První snahy o standardizaci těchto diagnostických zařízení vznikly v 80. letech minulého století v Kalifornii. Vládní organizace California Air Resources Board (CARB), která se zabývá ochranou životního prostředí, vydala nařízení, jež zavazovalo výrobce automobilů v USA k tomu, že jejich automobily prodané v roce 1988 a později v Kalifornii musely mít povinně diagnostický systém, aby mohlo být detekováno případné přílišné znečišťování okolí [3]. Tak vznikl první systém OBD (On-Board Diagnostic).

1.2 Sériová diagnostika

Sériovou diagnostiku automobilu provádíme zapojením speciálního testeru nebo notebooku, do konektoru definovaného normou OBD II. Tímto je zajištěno spojení mezi diagnostickým testerem a řídící jednotkou. Sériová diagnostika má následující funkce [1]:

- Resetování řídící jednotky do továrního nastavení.
- Čtení paměti závad.
- Vymazání paměti závad.
- Nastavení některých hodnot (například otáčky volnoběhu, omezovač otáček).

1.3 Paralelní diagnostika

Paralelní diagnostikou automobilu rozumíme přímé měření hodnot na jednotlivých čidlech či komponentách. Takto diagnostikovat lze například měřením napětí na pinech řídící jednotky. K měření se používá multimetr, v lepším případě osciloskop. Existují i specializované testery, které se připojí do obvodu mezi komponentu a řídící jednotku. Ty jsou ale velmi drahé, proto je vlastní většinou specializované servisy [1]. Paralelní diagnostika nemá žádnou z funkcí, kterou nabízí sériová diagnostika. Nelze pomocí ní číst v paměti závad, ani ji nijak upravovat. Této diagnostiky se využívá většinou u automobilů, které nepodporují připojení k sériové diagnostice.

1.4 Diagnostické standardy

Diagnostické standardy se vyvíjely odlišně v různých zemích podle toho, jaká měla daná země požadavky. V USA vznikla norma OBD, později OBD II, která se nakonec stala víceméně globální normou. Evropa přišla se svou normou EOBD, která definuje v mnoha směrech stejné standardy jako americká OBD II, obsahuje však přísnější emisní limity [4]. Standardizace byla nutná z důvodu kompatibility jednotlivých řídících jednotek automobilů s diagnostickými přístroji.

1.4.1 OBD I

OBD I byl poměrně jednoduchý systém, který sledoval data pouze z omezeného množství komponent, jako například ze systému EGR (Exhaust Gas Recirculation) a z řídící jednotky motoru. Tedy z komponent, které byly odpovědné za množství škodlivých látek vypouštěných ve výfukových plynech. OBD I nebyl nijak standardizován, rozdíly nebyly jen mezi různými značkami automobilů, ale i mezi jednotlivými modely stejného výrobce. Další slabostí systému bylo samotné hlášení závad. Některé poruchy neuměl systém odhalit vůbec, jako například nefunkční či odstraněný katalyzátor. Při výskytu závady se rozsvítila varovná kontrolka MIL (Malfunction Indicator Lamp) a následně tato kontrolka začala blikat kód dané chyby v Morseově abecedě. Kód chyby byl uložen do paměti [3]. OBD I byl krokem dobrým směrem, měl ale mnoho slabostí.

1.4.2 OBD II

Poučena z nedostatků, které obsahoval systém OBD I, přišla v roce 1989 organizace CARB (California Air Resources Board) s novým souborem norem pro OBD. Tak vznikl systém OBD II. Tímto diagnostickým systémem musela být povinně vybavena všechna vozidla prodávaná na území Kalifornie od roku 1995. OBD II standardizoval následující:

- diagnostickou zásuvku se 16 piny,
- funkce daného pinu zásuvky,
- chybové kódy,
- komunikační protokoly,
- terminologii.

Dalším pokrokem oproti OBD I je, že OBD II ukládá do paměti nejen kód chyby, která nastala, ale i podmínky, které panovaly v moment, kdy byla chyba detekována. Data, která obsahují tyto informace se souhrnně nazývají „Freeze frame“. Kvůli tomuto se změnil způsob signalizace kontrolky MIL. Tato kontrolka se rozsvítí vždy když je vypnutý motor, ale zapnuté zapalování. Pokud systém nezaznamená závadu,

po nastartování kontrolka zhasne. Kontrolka se také rozsvítí, pokud zaznamená závadu přímo související s emisními limity. Pokud závada nesouvisí přímo se zvýšením emisí, musí nastat minimálně dvakrát za stejných podmínek, aby byla rozsvícena kontrolka MIL [3].

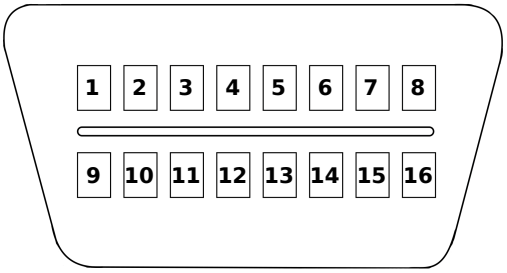
1.4.3 EOBD

V Evropě byla vytvořena norma EOBD (European On-Board Diagnostics). Všechna vozidla se zážehovým motorem musí mít od roku 2001 řídící jednotku kompatibilní s touto normou. Vozidla se vznětovým motorem musí tuto kompatibilitu poskytovat od roku 2003. Norma EOBD definuje stejné standardy jako OBD II, jen pod jiným názvem. Diagnostická data jsou zde dostupná v devíti režimech. Režimy 1-5 jsou určeny pro měření emisí, zbylé 4 režimy se používají pro diagnostiku motoru [1].

2 Komunikace s řídící jednotkou

2.1 OBD II konektor

Automobily, které splňují normu OBD II resp. EOBD, mají na výrobcem určeném místě konektor typu J1962 se 16 piny, který umožňuje komunikaci s řídící jednotkou vozu. Vzhled konektoru je na obrázku 2.1.



Obr. 2.1: Konektor OBD II.

Vozidla, která se řídí normou EOBD komunikují s řídící jednotkou pomocí technologie K-Line, která používá sedmý pin konektoru. Novější automobily využívají ke komunikaci sběrnici CAN, která využívá šestý a čtrnáctý pin. Vozidla z USA komunikují převážně pomocí protokolu SAE J1850 VPW, který využívá druhý pin, nebo pomocí SAE J1850 PWM, který je na druhém a desátém pinu. Piny 1,3,8,9,11,12,13, nemají v normě OBD II přesně specifikované využití [5]. Jednotliví výrobci si pak sami mohou určit, k čemu daný pin využijí. Kompletní popis využití jednotlivých pinů konektoru OBD II je v tabulce 2.1.

2.2 Funkce řídící jednotky

Řídící jednotka je palubní počítač, který snímá hodnoty nejrůznějších veličin z množství čidel. Tyto hodnoty porovnává s referenčními hodnotami, které má uložené ve své paměti „Flash“. Pokud data z čidel odpovídají referenčním hodnotám, je všechno v pořádku. Řídící jednotka poté zahrnuje tyto informace do dalších výpočtů, například složení směsi paliva nebo řízení zážehu. V případě, že hodnoty, které řídící jednotka obdrží ze senzoru, nejsou v souladu s hodnotami v paměti Flash, označí řídící jednotka tento senzor jako vadný. Toto provede uložením příslušného kódu do paměti EEPROM (Electrically Erasable Programmable Read-Only Memory).

Pokud je čidlo označeno jako vadné, přestane řídící jednotka zahrnovat hodnoty z tohoto čidla do svých dalších výpočtů. Nahradí je buď referenčními hodnotami nebo pro získání potřebných hodnot použije jiné čidlo, pokud je to možné. Aby řídící

Tab. 2.1: Přehled obsazení pinů konektoru OBD II [5].

Číslo pinu	**Popis**
1	Nespecifikováno
2	J1850 PWM + nebo J1850 VPW +
3	Nespecifikováno
4	Kostra (šasi) vozidla
5	Signálová kostra
6	CAN High
7	K-line komunikační linka
8	Nespecifikováno
9	Nespecifikováno
10	J1850 PWM -
11	Nespecifikováno
12	Nespecifikováno
13	Nespecifikováno
14	CAN Low
15	L-line inicializační linka nebo 2. K-line
16	Palubní napětí +12V (baterie)

jednotka opět začala využívat data z daného čidla, je potřeba vymazat z paměti EEPROM chybové hlášení [1].

2.3 Navázání spojení s řídící jednotkou

Po připojení testeru do konektoru, vysílá tester adresní bajt. Řídící jednotka odpoví sekvencí bajtů, ve které definuje, který protokol bude pro komunikaci použit. Tester následně tuto skutečnost řídící jednotce potvrdí. Po nastavení protokolu pošle řídící jednotka svou identifikaci (například VIN kód, objednací číslo, konfigurační kód paměti EEPROM).

Po tomto úvodním sledů akcí přechází komunikace do tzv. „Idle“ režimu. V něm řídící jednotka a tester komunikují jen na příkaz obsluhy testeru [1].

2.4 Diagnostické režimy

Norma SAE J1979 definuje režimy, ve kterých je diagnostický tester schopný pracovat. První bajt přijatých dat obsahuje informaci o tom, jaký mód je použit. Teoreticky by měly všechny testery být schopny interpretovat stejné informace, v praxi

tomu ale tak není. Nejvíce problematické je zobrazování tzv. PID kódů. Diagnostické módy jsou přehledně vypsány v tabulce 2.2 [4, 6].

Tab. 2.2: Diagnostické módy.

Mód 1	Měřené hodnoty
Mód 1-01	Readiness – stav emisní připravenosti
Mód 2	Freeze frame
Mód 3	Paměť závad
Mód 4	Vymazání závady
Mód 5	Test lambda sond
Mód 6	Průběžný test
Mód 7	Paměť sporadických závad
Mód 8	Test akčních členů
Mód 9	Informace o vozidle
Mód 10	Trvalé závady

2.5 PID kódy

PID (Parameter IDs) jsou hexadecimální kódy sloužící ke komunikaci s řídící jednotkou podporující OBD II. Skládají se ze dvou částí. První částí je mód, ve kterém diagnostika pracuje. V druhé části je pak samotná hodnota PID. Příkladem PID kódu může být například `01 0C`. To znamená, že diagnostika pracuje v módu 1 a 0C je dotaz na hodnotu otáček motoru. Odpověď bude v tomto případě obsahovat dvě proměnné, z nichž se vypočítá výsledná hodnota otáček motoru, podle vzorce definovaného v normě SAE J1979. Komunikace mezi řídící jednotkou a testerem probíhá pomocí těchto kódů. Obsluha testeru zadá daný kód a řídící jednotka odpoví kódem, který obsahuje požadovaná data [7, 8].

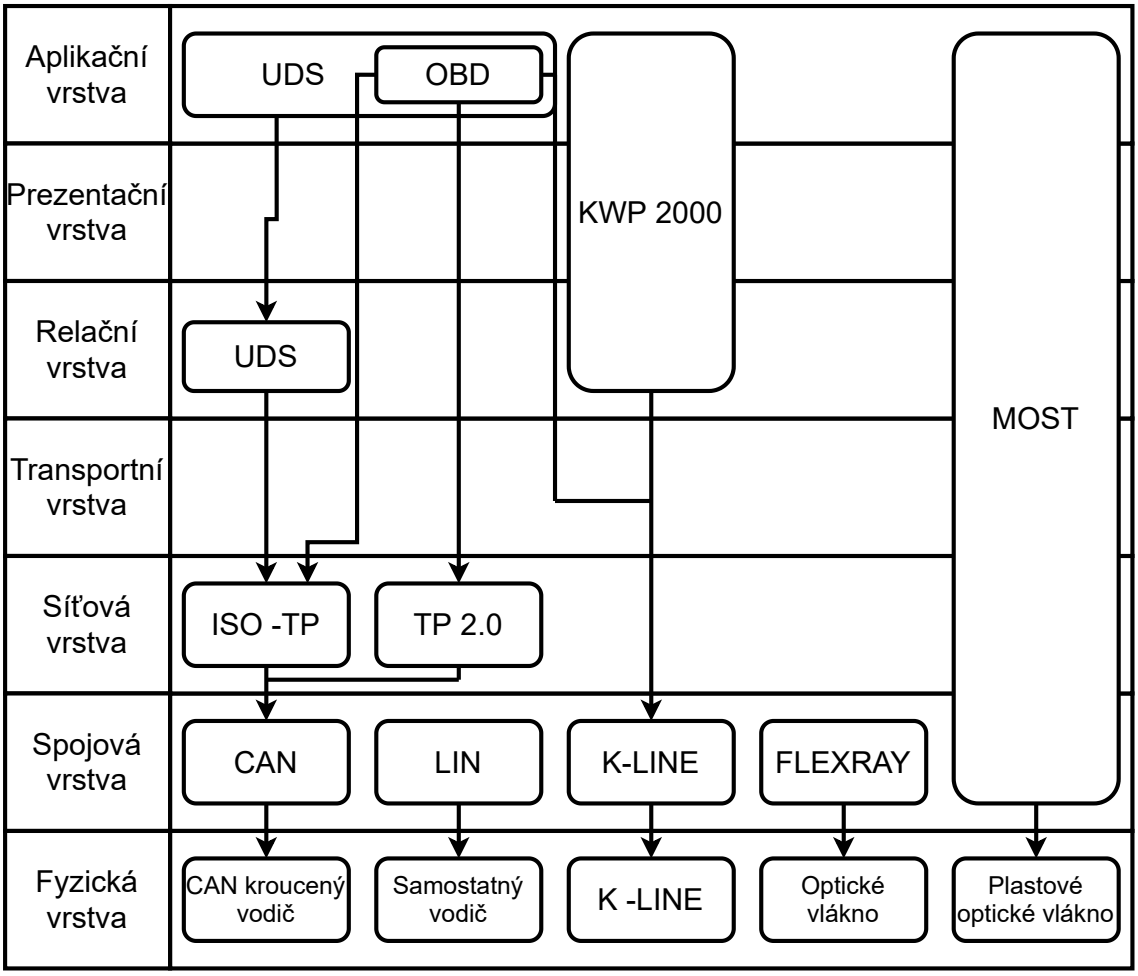
2.6 Komunikační protokoly

Aby byla možná komunikace mezi testerem a řídící jednotkou po připojení do konektoru OBD II, musí existovat jakýsi komunikační jazyk, který bude ovládat jak řídící jednotka, tak tester. Tento jazyk se nazývá protokol. Protokol definuje všechny náležitosti komunikace jako jsou [1]:

- význam klíčových slov,
- rychlost posílání bajtů,

- doba čekání na příjem bajtu,
- doba za kterou se pošle další bajt.

Pokud není dodržena některá z výše uvedených podmínek, je potřeba spojení navázat znovu. Na obrázku 2.2 jsou uvedeny nejobvyklejší protokoly, s nimiž se můžeme setkat v automobilech promítnuté do síťového modelu ISO/OSI [9]. Některé z těchto protokolů budou podrobněji popsány v následujících odstavcích.



Obr. 2.2: Nejobvyklejší protokoly používané v automobilech

2.6.1 SAE J2248 CAN

Tento protokol funguje na sběrnici CAN (Controller Area Network). Sběrnice CAN byla vyvinuta firmou Bosch v 80. letech minulého století. Její užívání v sériové výrobě začalo v 90. letech a od roku 2001 je povinně použita v evropských vozech. V USA byla standardizována v roce 2008. Sběrnice CAN je tvořena dvěma vodiči, označenými jako CAN High a CAN Low. Ty jsou vyvedeny na piny 6 a 14 konektoru

OBD II. Zprávy, které prochází sběrnicí neobsahují informaci o cílovém uzlu. Jsou zaslány všem uzlům ve sběrnici.

CAN je jednoduchý sériový protokol, který umožňuje řízení systému v reálném čase. Z tohoto důvodu se nevyužívá jen v automobilovém průmyslu, ale i v dalších průmyslových odvětvích, kde je potřeba tohoto řízení. Jedná se o tzv. multi-master protokol, což znamená, že jednotlivé uzly mohou řídit další uzly. Není zde jeden nadřazený uzel, který by řídil všechny ostatní. Tento faktor zvyšuje spolehlivost celého systému. Maximální přenosová rychlost tohoto protokolu je až 1 Mb/s. S rostoucí délkou vedení ale rychlost přenosu strmě klesá. Nejvyšší rychlosti přenosu lze dosáhnout pouze na krátké vzdálenosti, zhruba do 40 m, což je v automobilech dodrženo [1].

2.6.2 K-Line

K-Line nebo také ISO 9141-2 je protokol používaný nejvíce v evropských automobilech. Používá pin 7, případně ještě pin 15 na konektoru OBD II.Na rozdíl od paketů protokolu CAN mají pakety protokolu K-Line adresu zdroje i adresu příjemce [7].

2.6.3 KWP 2000

Keyword protokol, známý také jako KWP 200, definovaný v ISO 14230, je častým protokolem u aut z USA vyrobených po roce 2003. Jde o vylepšenou verzi protokolu K-Line. Jeho zprávy mají maximální délku 255 bajtů. Používá pin 7 konektoru OBD II. Existují dvě verze tohoto protokolu [7]:

- ISO 14230-4 KWP (5 baud init, 10,4 kb/s),
- ISO 14230-4 KWP (fast init, 10,4 kb/s).

Přičemž druhá verze umožňuje rychlejší navázání komunikace mezi řídící jednotkou a testerem.

2.6.4 SAE J1850 PWM

SAE J1850 PWM je prvním ze dvou variací protokolu SAE J1850. Stejně jako u protokolu CAN je zde použito diferenciální kódování, to znamená, že logické hodnoty 1 a 0 jsou reprezentovány jako rozdíl dvou napětí. K přenosu informací je použito pulzně šířkově modulace. Anglicky Pulse Width Modulation (PWM). Odtud pochází označení protokolu. PWM používají hlavně automobily značky Ford. Tento protokol používá piny 2 a 10. Jeho maximální přenosová rychlost je 41,6 kb/s [7].

2.6.5 SAE J1850 VPW

Druhým protokolem typu SAE J1850 je SAE J1850 VPW. Zkratka VPW (Variable Pulse Width) znamená „variabilní šířka pulzu“. To napovídá, že u tohoto protokolu je stejně jako SAE J1850 PWM, použito pulzně šířkové modulace, avšak s variabilní šířkou pulzu. Tento protokol používají automobily z koncernu General Motors a vozy Chrysler. Konektor OBD II má pro protokol SAE J1850 PWM vyhrazen pin 2. Maximální přenosová rychlost tohoto protokolu je 10,4 kb/s [6, 7].

2.6.6 Local Interconnect Network Protocol

Local Interconnect Network Protocol (LIN) je nejjednodušší a nejlevnější na implementaci z protokolů, použitých v automobilech [7]. Byl vytvořen, aby doplňoval protokol CAN. Má jediný uzel typu master, který je zodpovědný za veškerý přenos. Může mít až 16 podřízených uzlů, které většinou pouze naslouchají nadřízenému uzlu. Maximální přenosová rychlost tohoto protokolu je 20 kb/s. LIN nemá přímo vyhrazený pin na konektoru OBD II, někdy bývá ale použit místo CAN protokolu.

2.6.7 FlexRay

Protokol FlexRay umožňuje velmi rychlou komunikaci o rychlosti až 10 Mb/s. Oproti protokolům CAN a LIN je rychlejší a odolnější vůči rušení elektromagnetickým polem. FlexRay lze použít ve sběrnicové topologii nebo v topologii typu hvězda. Tento protokol je z pohledu nákladů nejdražší na použití, proto se používá spíše výjimečně [7].

2.6.8 UDS

Hlavní funkcí protokolu UDS (Unified Diagnostic Services) je komunikovat se všemi řídícími jednotkami, které automobil obsahuje, zajišťovat údržbu tím, že bude hledat chyby, aktualizovat firmware atd. Při testování funguje jako protokol typu klient-server. Tester (klient) zasílá požadavky, které mohou být přesměrovány na jednu nebo více řídících jednotek (servery). Řídící jednotka poté odpovídá kladnými nebo zápornými odpověďmi [10].

2.7 Chybové kódy DTC

Řídící jednotka ukládá záznamy o chybách jako kódy DTC (Diagnostic Trouble Codes). Tyto záznamy jsou uloženy do různých typů pamětí, podle toho, o jakou chybu se jedná. Kódy, které informují o méně závažné závadě se ukládají do paměti

RAM (Random Access Memory). V případě, že dojde k odpojení baterie, dojde ke smazání všech chybových kódů uložených v paměti RAM. Závažnější chyby jsou uloženy do paměti, která není závislá na stálém zdroji elektrické energie.

Z hlediska závažnosti, lze závady rozdělit na třídy A,B,C,D [1, 7]. Závady třídy A znamenají, že došlo k hrubému porušení emisních limitů a ihned se rozsvítí kontrolka MIL (Malfunction Indicator Lamp). V případě, že se objeví závady třídy B, které nesouvisí s emisními limity, dojde k rozsvícení kontrolky MIL pouze při opakovaném výskytu této chyby. Při výskytu chyb třídy C nebo D nedojde k rozsvícení kontrolky MIL.

Při ukládání DTC uloží řídící jednotka také data z dalších čidel automobilu, které nějakým způsobem souvisí s motorem. Ideální by bylo uložit tato data přesně v okamžiku, kdy nastane chyba. Reálně ale dojde k uložení zhruba s pětisekundovým zpožděním. Těmto datům se souhrnně říká „Freeze Frame Data“ a obsahují následující [7]:

- DTC,
- zátěž motoru,
- otáčky motoru,
- teplotu motoru,
- složení palivové směsi,
- tlak vzduchu,
- operační mód (otevřený nebo uzavřený cyklus),
- rychlost automobilu,
- pozice plynového pedálu.

DTC jsou jsou alfanumerické kódy s délkou 5 znaků. První znak je písmeno, určující část automobilu, která uložila kód. Hodnoty prvního znaku mohou být P, B, C, U. P (Powertrain) značí motor, převodovku a všechny chyby spojené s množstvím emisí. B (Body) reprezentuje karoserii a s ní spojené závady jako jsou například světla nebo klimatizace. C (Chassis) je použito u kódů, které charakterizují závady na podvozku. U (User Network) jsou kódy indikující s poruchami v komunikační síti automobilu.

Druhý znak může nabývat hodnot 0, 1, 2, 3. V moderních automobilech jsou hodnoty 0, 2, 3 standardizovány podle SAE. Hodnota 1 je definována výrobcem. Ve starších modelech se můžeme setkat s tím, že hodnota 3 je také definována výrobcem.

Hodnota třetího znaku je závislá na hodnotě prvního. Definuje totiž podřazený systém, ve kterém nastala chyba.

Čtvrtý a pátý znak blíže definují nastalou závadu a jejich hodnoty se liší v závislosti na hodnotách předchozích znaků.

Každý kód DTC je v binární podobě reprezentován dvěma bajty, tj. šestnácti bity. V tabulce 2.4 je názorná ukázka zápisu kódu P0477 v hexadecimálním, bi-

Tab. 2.3: Složení DTC kódů [7].

Pozice znaku	**Význam**
1	P (Powertrain), B (Body), C (Chassis), U (User Network)
2	0, 2, 3 (SAE standard) 1,3 (Definováno výrobcem)
3	Podřazený systém systému definovanému na pozici 1
4	Bližší specifikace závady
5	Bližší specifikace závady

nárním a DTC formátu. Tento kód značí, že senzor tlaku ve výfukovém potrubí nedodává správné hodnoty.

Tab. 2.4: Binární a hexadecimální skladba DTC [7].

Formát	**1. bajt**			**2. bajt**	
Hex	0x0		0x4	0x7	0x7
Bin	00	00	0100	0111	0111
DTC	P	0	4	7	7

DTC kódy se obvykle vymažou samy, pokud závada, kterou indikují, nenastane za stejných podmínek znovu. Stejnými podmínkami se rozumí [7]:

- Otáčky motoru jsou v rozmezí ± 375 RPM od hodnoty, která byla zaznamenána poprvé.
- Zatížení motoru je maximálně o ± 10 % jiné, než při prvním výskytu chyby.
- Teplota motoru je stejná, jako při prvním záznamu chyby.

Pokud se kódy nevymažou samy, je možné je vymazat pomocí diagnostického přístroje. Některé méně závažné chyby, uložené v paměti RAM jsou vymazány po odpojení autobaterie. DTC indikující závažnější poruchy jsou odstraněny pouze potom, co je porucha opravena. Toto slouží jako ochrana před nepoctivými mechaniky, kteří by pouze vymazali DTC z paměti, ale závadu neopravili.

3 Rozbor problematiky a návrh řešení

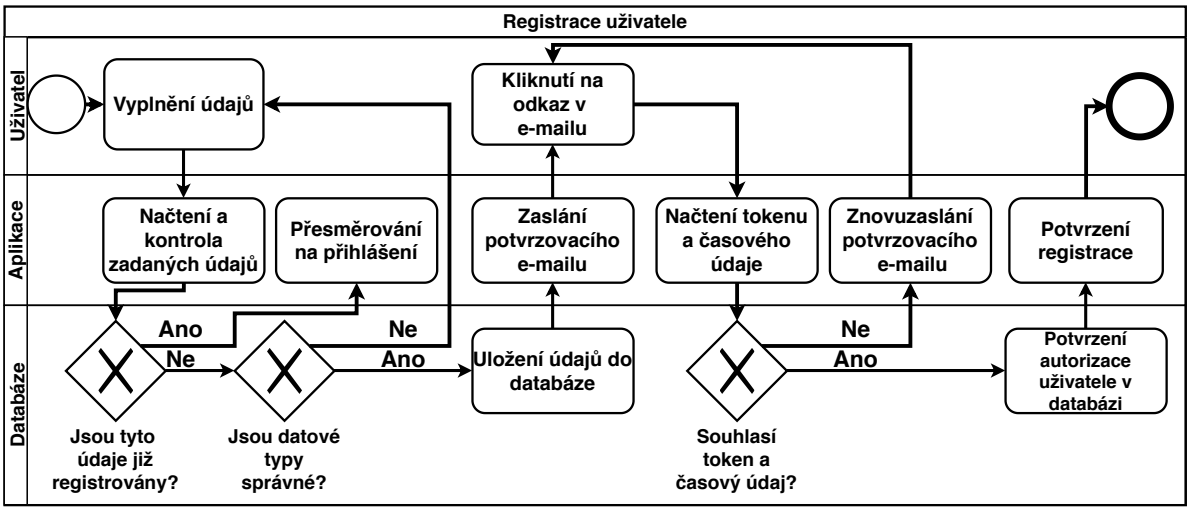
Vývoj aplikace je komplikovaný proces, který je vhodné před započetím důkladně promyslet. Tento proces vyžaduje komplexní rozbor problematiky daného projektu. Je třeba si ujasnit, jaké bude logické pořadí procesů, jaká bude struktura jednotlivých částí, jaké možné situace mohou nastat, jak se tyto situace budou řešit a jaké technologie budou pro vývoj použity.

3.1 BPMN diagramy

BPMN (Business Process Model and Notation) je modelovací jazyk, sloužící především pro modelování podnikových procesů. Využívá se ale i při vytváření modelů softwarových řešení. Tyto modely slouží pro přehlednou ilustraci jednotlivých procesů a osvětlení principů jejich fungování. Primárním cílem je vytvoření modelů, kterým budou rozumět jak IT odborníci, tak manažeři a ostatní zaměstnanci, kteří s těmito diagramy přijdou do styku [12].

3.1.1 Diagram registrace uživatele

Aby mohl uživatel využívat služeb této aplikace, je nutné, aby se zaregistroval do systému. Průběh této registrace je přehledně rozkreslen na obrázku 3.1.



Obr. 3.1: BPMN diagram registrace uživatele.

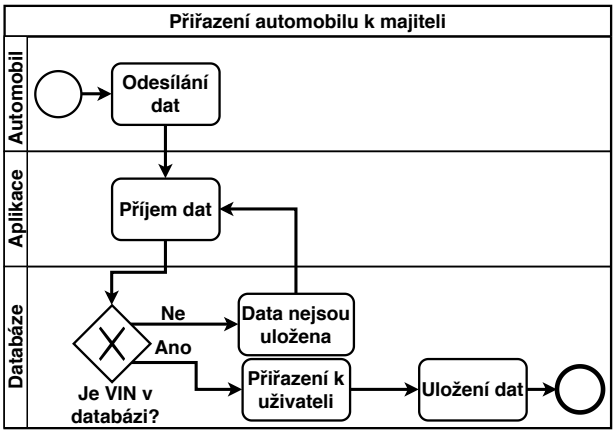
Tento BPMN diagram se skládá ze tří částí: uživatel, aplikace a databáze. V každé z těchto částí jsou definovány operace, které daná entita provádí. Start je symbolizován kruhem s tenkým okrajem, konec je symbolizován kruhem s širším okrajem.

Jednotlivé akce jsou znázorněny obdélníky se zaoblenými rohy. Rozhodování je znázorněno kosočtverci s křížkem uvnitř.

Uživatel nejdříve zadá své osobní údaje, jako jsou jméno, příjmení, uživatelské jméno, heslo a e-mail, do předem připraveného formuláře na webových stránkách. Následně proběhne načtení těchto údajů a kontrola duplicity. Pokud je e-mail, který uživatel zadá, shodný s e-mailem, který již v databázi je, dojde k přesměrování uživatele na stránku s přihlašovacím formulářem. V případě, že je e-mail neznámý, dojde k ověření správnosti zadaných informací vzhledem k použitým datovým typům. Dále je také zkontrolováno, zda uživatel vyplnil všechna pole. Správné zadaní informací vede k uložení uživatele do databáze. Kromě uživatelem zadaných informací, je k uživateli uložen také jeho status ověření, který nabývá pouze hodnot „true“ nebo „false“.

Do jiné tabulky v databázi se uloží náhodně vygenerovaná posloupnost znaků („token“), čas jejího vytvoření a údaj o tom, kterému uživateli tento token náleží. Následně je uživateli odeslán potvrzovací e-mail, který obsahuje odkaz s tokenem. Kliknutím na odkaz je načten token i časový údaj. Pokud token souhlasí a je starý maximálně hodinu, je uživatel ověřen, jeho účet je nyní aktivní a uživatel je schopný se do systému přihlásit. V případě, uživatel na odkaz neklikne nebo není splněna jedna z výše zmíněných podmínek, potvrzení neproběhne a uživatel není schopen se přihlásit.

3.1.2 Diagram přiřazení automobilu k uživateli



Obr. 3.2: BPMN diagram přiřazení automobilu majiteli.

BPMN diagram přiřazení automobilu k majiteli je složen ze tří sekcí: automobil, aplikace a databáze. Nastiňuje komunikaci mezi těmito entitami a je na obrázku 3.2. Po připojení jednotky do zásuvky OBD II bude automobil posílat data aplikaci

v nastavených časových intervalech. Aby aplikace data přijala a uložila, je nutné, aby byl automobil přiřazen k majiteli. Přiřazení bude možné pomocí VIN (Vehicle Identification Number) kódu, který je pro každý automobil jedinečný. VIN kód bude muset nejdříve zadat uživatel do systému, následné přiřazení automobilu se provede automaticky pomocí tohoto kódu. Pokud je VIN kód automobilu vysílajícího data pro databázi neznámý, nejsou data přijatá od tohoto automobilu uložena. V případě, že VIN kód automobilu posílajícího data je v databázi, jsou přijatá data přiřazena k tomuto automobilu a uložena do databáze.

3.2 Entitně relační diagram databáze

Entitně relační diagramy se používají pro popis objektů (entit) a jejich vztahů s ostatními objekty. Díky použití grafického znázornění jsou tyto diagramy velmi jednoduše srozumitelné. V tomto případě jsou jednotlivými entitami tabulky vytvořené v databázi. V databázi typu SQL, jež je použita v tomto projektu jsou vytvořeny následující tabulky:

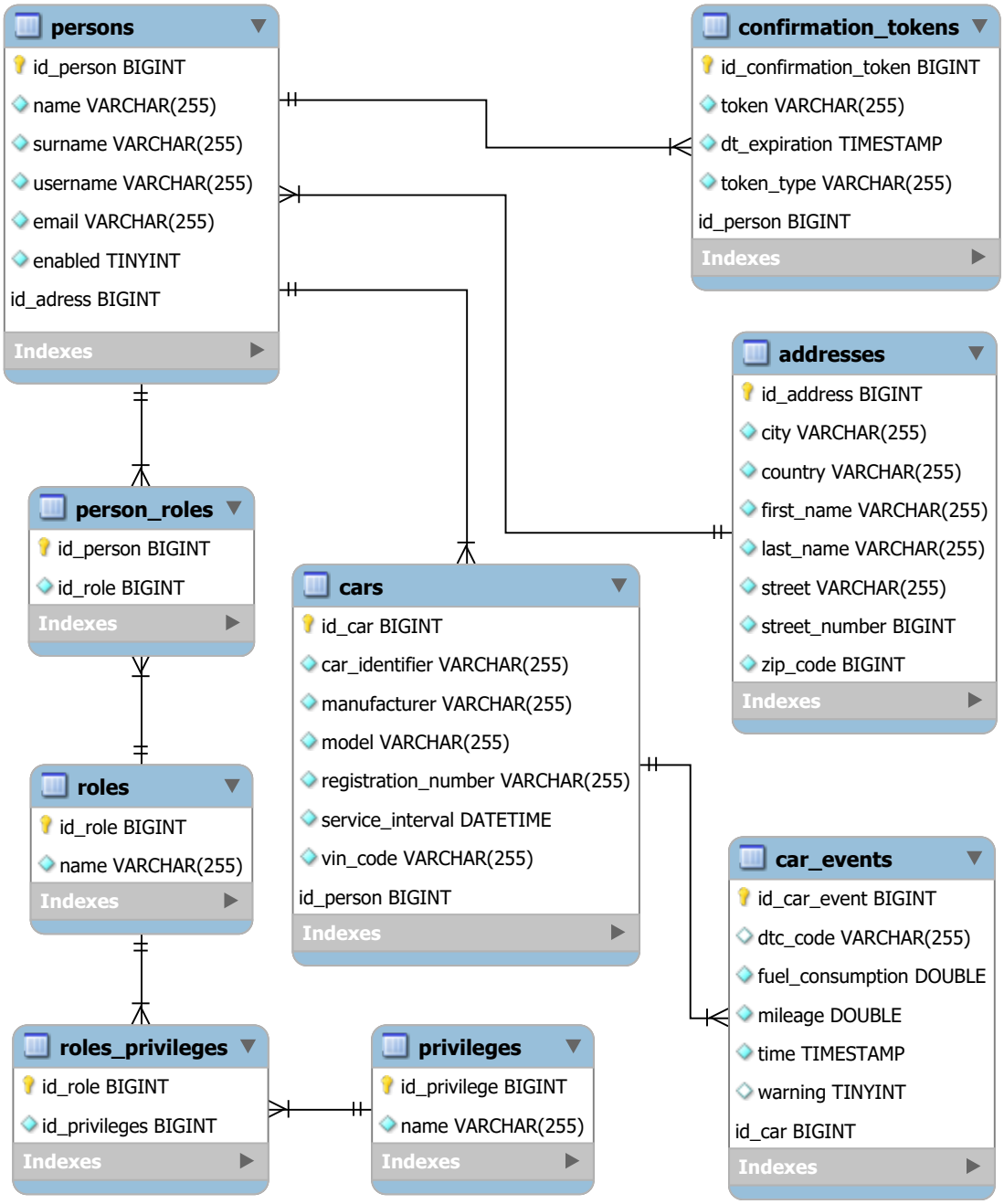
- `persons` – Obsahuje informace o uživatelích.
- `cars` – Obsahuje informace o automobilech.
- `addresses` – Obsahuje informace o korespondenčních adresách uživatelů.
- `confirmation_tokens` – Obsahuje potvrzovací tokeny.
- `car_events` – Obsahuje hlášení z automobilů.
- `person_roles` – Pomocná tabulka pro přiřazení rolí uživatelům.
- `roles` – Obsahuje role uživatelů.
- `roles_privileges` – Pomocná tabulka pro přiřazení práv jednotlivým rolím.
- `privileges` – Obsahuje informace o právech.

V tabulce jsou vždy uvedeny atributy, které jednotlivé entity mají. Primární klíč každé tabulky je označen symbolem zlatého klíče. Povinné atributy jsou vyznačeny modrým plným kosočtvercem, nepovinné atributy mají vedle názvu kosočtverec s modrým okrajem bez výplně. U atributů je určen datový typ, který je použit pro jejich uložení. V případě, že jsou atributy uloženy jako řetězec znaků („varchar“), je v závorce uvedena ještě délka daného řetězce.

Vztahy mezi entitami se podle násobnosti (kardinality) dělí na [13]:

- 1:1 – Na každé straně vztahu vystupuje pouze jeden objekt dané entity.
- 1:N – Na jedné straně je jeden objekt, který má vztah s více objekty na druhé straně.
- M:N – Na obou stranách vystupuje více objektů.

V entitně relačním diagramu na obrázku 3.3 jsou vztahy pouze 1:N resp. N:1, přičemž strana, u které je symbol šipky, je stranou s více objekty. Naopak strana, u níž jsou dvě svislé čáry je stranou s jediným objektem. V praxi to pak znamená, že



Obr. 3.3: Entitně relační diagram databáze.

jedna osoba může mít více automobilů a potvrzovacích tokenů nutných pro potvrzení registrace. Jeden automobil může mít více hlášení o stavu vozidla (entit `car_events`) a tak dále.

3.3 Použité nástroje

Pro vytvoření této aplikace bylo využito poměrně velkého množství technologií. Hlavní z nich jsou velmi stručně představeny v následujících odstavcích.

3.3.1 Java

Programovací jazyk Java, původně pojmenovaný Oak, byl vyvinut v roce 1991 v USA týmem programátorů, který vedl James Gosling. Hlavní motivací pro vývoj tohoto jazyka byla potřeba mít programovací jazyk, který není závislý na platformě a je možné ho použít i ve spotřební elektronice a drobných zařízeních. Druhým důležitým faktorem, který ovlivnil vznik jazyka Java, byl turbulentní vývoj internetu, který pro svá různá zařízení potřeboval právě software nezávislý na platformě, podle slavného motta jazyka Java: „Write once, run anywhere“. Hlavními výhodami tohoto jazyka jsou [14]:

- Jednoduchost – Jazyk Java zdědil část své syntaxe z jazyků C resp. C++, proto je jednoduché přejít na tento jazyk, pokud již programátor má zkušenosti s objektově orientovaným programováním z jiných jazyků.
- Nezávislost na platformě – Programy napsané v jazyce Java by měly fungovat na jakémkoliv zařízení, bez ohledu na hardware či software díky dvojitému překladu, který nejdříve přeloží kód do tzv. „bytecode“ a poté do strojového kódu.
- Robustnost – Java obsahuje tzv. „garbage collector“, který sám provádí správu paměti a automaticky dealokuje paměť, která již není využívána.
- Podpora vícevláknových procesů – Podporuje vývoj programů, které provádějí více než jednu operaci zároveň.
- Vysoká výkonnost – Díky překladu do „bytecode“, který je velmi dobře optimalizovaný je dosaženo vysoké výkonnosti.

3.3.2 Spring

Spring je open source framework založený na jazyce Java. Framework obecně je sbírka knihoven, které mají za úkol zjednodušit vývoj programu již předdefinovanými metodami, konfiguracemi nebo řešeními problémů. Spring byl vytvořen v roce 2002 Rodem Johnsonem. Jeho primárním cílem bylo zvýšit produktivitu vývojářů tím, že jim poskytne sadu knihoven, která vyřeší část problémů za ně. Spring se stal populárním hlavně díky jednoduchému vkládání závislostí (dependency injection) a své modularitě. Právě jeho modularita je výraznou výhodou, která zajišťuje široké možnosti použití tohoto frameworku v různých aplikacích. Základními moduly frameworku Spring jsou [15]:

- Spring Core – Základní modul, který umožňuje vkládání závislostí a základní webové funkce.
- Spring Data – Modul, který zajišťuje jednoduchou správu dat aplikace a propojení s databází.
- Spring Security – Bezpečnostní modul, přinášející možnosti autentizace a autorizace.
- Spring Cloud – Modul pro vývoj distribuovaných systémů.
- Spring Web MCV – Modul podporující vývoj webových aplikací.

3.3.3 REST API

Obecně řečeno API (Application Programming Interface) je nástroj, který umožňuje aplikacím komunikovat se serverem. API naslouchá požadavkům klienta a odpovídá na ně. Jako REST API se označuje API, které vyhovuje architektuře REST. REST (Representational State Transfer) byl vytvořen v roce 2000 jako součást disertační práce Roye Fieldinga [16]. Využívá základních HTTP (Hypertext Transfer Protocol) metod jako jsou:

- POST – Slouží k vytvoření dat.
- GET – Slouží k získání požadovaných dat.
- PUT – Slouží k nahrazení stávajících dat novými.
- DELETE – Slouží k odstranění dat.
- PATCH – Slouží k úpravě stávajících dat.
- HEAD – Je metoda se stejným účelem jako metoda GET, ale server nesmí poslat „message body“ jako odpověď.
- OPTION – Představuje požadavek na informace o možnostech komunikace.

3.3.4 Hibernate

Hibernate je jednou z implementací JPA (Java Persistence API). Jde o framework, který umožňuje objektově relační mapování. To znamená, že dovoluje jednoduše pracovat s daty z relační databáze ve formě Java objektů. V praxi to znamená, že není nutné psát všechny SQL příkazy, stačí použít anotace předdefinované v tomto frameworku [17]. Tento faktor výrazně usnadňuje práci s databázemi. Mezi Hibernate anotace použité v tomto projektu patří například:

- `@Entity` – Definuje, která třída, rozhraní nebo enum, je entitou.
- `@Table` – Umožňuje specifikovat detaily tabulky, do které bude entita uložena.
- `@Column` – Slouží ke specifikaci sloupce, do kterého bude namapován atribut entity.
- `@Id` – Definuje, který atribut je primárním klíčem.
- `@ManyToOne`, resp. `@OneToMany` – Vytvoří vztah N:1, resp. 1:N mezi entitami.

3.3.5 HTML a CSS

Přestože se jedná o dvě různé technologie, v reálném nasazení jsou spolu HTML a CSS neodmyslitelně spjaty. Proto budou i v této práci krátce popsány společně v jedné podkapitole. HTML (HyperText Markup Language) je programovací jazyk, sloužící pro tvorbu webových stránek, který vznikl spolu s technologií WWW(World Wide Web) v roce 1990. Veřejně spolu s dokumentací byl ale oficiálně vydán o rok později [18]. Od svého počátku prodělal velké změny a momentálně se používá verze HTML5. Jazyk HTML je postaven na použití značek (tagů) pro jednotlivé bloky. Dovnitř těchto tagů lze poté umístit text, grafiku nebo například multimédia. Při tvorbě tohoto jazyka nebyl kladen důraz na výsledný vzhled stránky, ale na její funkčnost. Později byly do tohoto jazyka implementovány atributy, pomocí níž bylo možné vytvořit poněkud přívětivější vzhled. Nevýhodou tohoto přístupu ale bylo, že se kód stával mnohem méně přehledným a celková struktura stránky nebyla v této změti kódu na první pohled patrná.

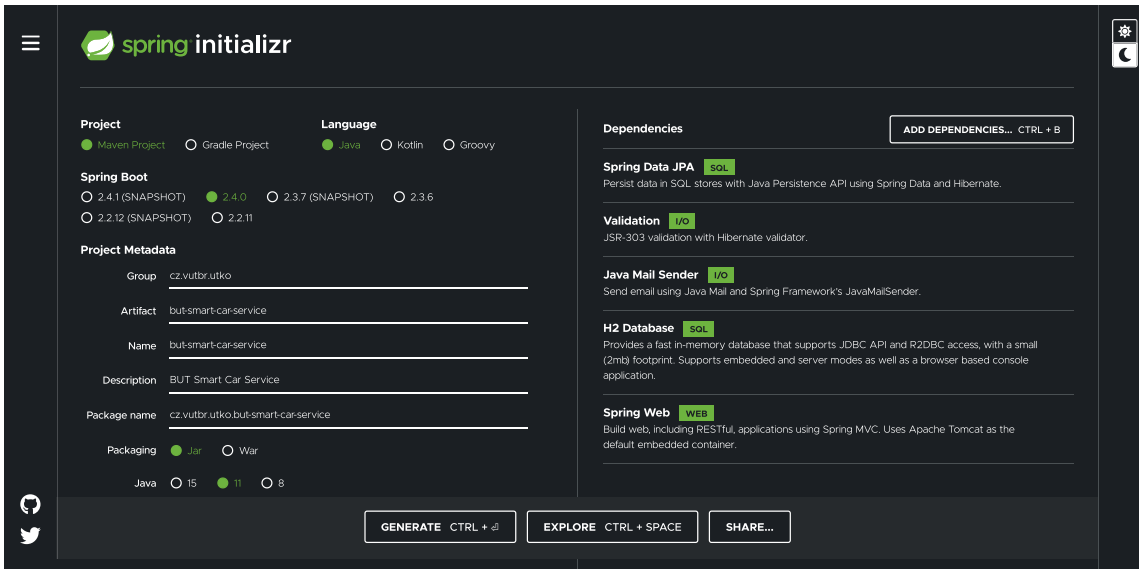
Tyto nevýhody byly odstraněny se vznikem CSS (Cascading Style Sheets). CSS je programovací jazyk sloužící k tvorbě stylů pro strukturované dokumenty, jako jsou například HTML nebo nebo XML (Extensible Markup Language) dokumenty. Hlavní výhodou jeho použití je, že je od sebe oddělen kód pro obsah a kód pro styl webových stránek. S pomocí CSS je možné vytvořit celkové rozložení stránky, vzhled pro jednotlivé elementy, ze kterých se stránka skládá či styly písma [19].

4 Vývoj aplikace

Pro vývoj aplikací v jazyce Java s podporou frameworku Spring se obecně používají dvě hlavní vývojová prostředí. Eclipse, vyvinuté společností IBM a IntelliJ od společnosti JetBrains. Obě mají své klady a zápory, v poslední době se ale větší oblibě mezi vývojáři těší IntelliJ [20]. Hlavními výhodami jsou více intuitivní ovládání, lepší automatické doplňování kódu a jednoduché refaktorování kódu. Pro studenty je také možné využívat verzi IntelliJ Ultimate, která v sobě obsahuje také prvky z vývojového prostředí WebStorm, které bylo vytvořeno pro programování v jazycích JavaScript, HTML a CSS. Tato vlastnost byla velmi přínosnou při tvorbě uživatelského rozhraní aplikace. Z těchto důvodu bylo IntelliJ vybráno jako IDE (Integrated Development Enviroment) pro tento projekt.

4.1 Založení projektu

Založení projektu, který pracuje s frameworkem Spring je výhodné provést pomocí nástroje Spring Initializr, což je generátor čistého projektu, který sám vytvoří základní strukturu projektu, popřípadě přidá podporu dalších zvolených technologií. Tento generátor je možné použít buď přímo ve vývojovém prostředí IntelliJ, nebo je dostupný na adrese `https://start.spring.io`. Vzhled tohoto webového generátoru je na obrázku 4.1. V nástroji Spring Initializr je možné nastavit všechny



Obr. 4.1: Založení projektu pomocí Spring Initializr.

potřebné atributy projektu. V tomto případě byl zvolen druh projektu Maven, programovací jazyk Java, balíčkování Jar a verze Java 11. V sekci „Dependencies“ je

pak možno přidat závislosti projektu na externích knihovnách. V tomto případě byly zvoleny závislosti:

- Spring Web – Základní balíček k tvorbě webů.
- Spring Data JPA – Slouží k práci s daty, přidává do projektu podporu frameworku Hibernate.
- H2 Database – Jednoduchý databázový systém.
- Validation – Obsahuje nástroje pro validaci dat vkládaných do databáze.
- Java Mail Sender – Obsahuje metody pro sestavení e-mailu a jeho odeslání.
- Thymeleaf – Nástroj pro práci s HTML, XML a CSS.

V případě, že by některá ze závislostí nebyla přidána při generování projektu, je později možné potřebnou závislost přidat později do souboru `pom.xml`. Po kliknutí na tlačítko „GENERATE“ dojde ke stažení čistého projektu v archivu zip. Po rozbalení je možné s ním začít pracovat. Takto vygenerovaný projekt již obsahuje základní strukturu projektu. To výrazně usnadňuje práci, protože není třeba ručně tvořit tuto hierarchii složek a souborů. Dále je také již vytvořena třída main a soubor `pom.xml`. POM (Project Object Model) je základní částí projektu Maven. Nachází se v kořenové složce projektu a obsahuje všechny informace o projektu a jeho konfiguraci.

4.2 Architektura aplikace

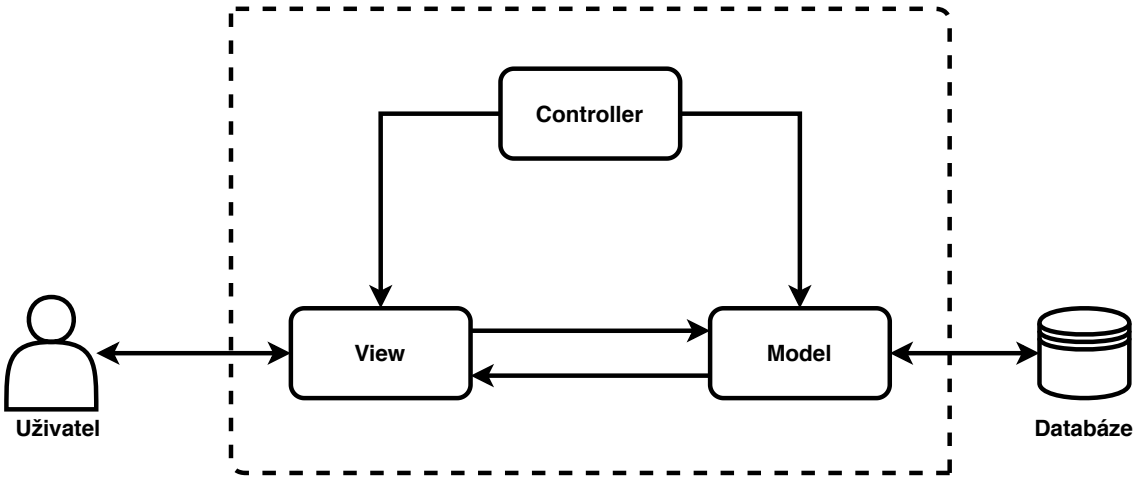
Jelikož se tento projekt skládá jak z uživatelského rozhraní (Frontend), tak ze serverové části (Backend), byla pro tuto aplikaci zvolena architektura, která je spojením dvou různých přístupů. Pro backend byla zvolena poměrně robustní N–vrstvá architektura, zatímco frontend byl do projektu zakomponována pomocí MVC architektury.

4.2.1 Model-View-Controller

MVC (Model-View-Controller) je architektura, která rozděluje aplikaci na tři hlavní logické komponenty, přičemž každá z těchto komponent se stará o specifickou část aplikační logiky. Tyto komponenty jsou [21]:

- Model – Zajišťuje veškerou práci s daty.
- View – Zajišťuje zobrazení a uživatelské rozhraní.
- Controller – Přijímá příchozí požadavky a podle nich řídí výše zmíněné komponenty.

Na obrázku 4.2 je znázorněno fungování architektury MVC. Uživatel pracuje s komponentou `View`, kterou může být například stránka s určitými informacemi. Pokud uživatel klikne na odkaz, který vede na požadovanou stránku, komponenta



Obr. 4.2: Architetura MVC.

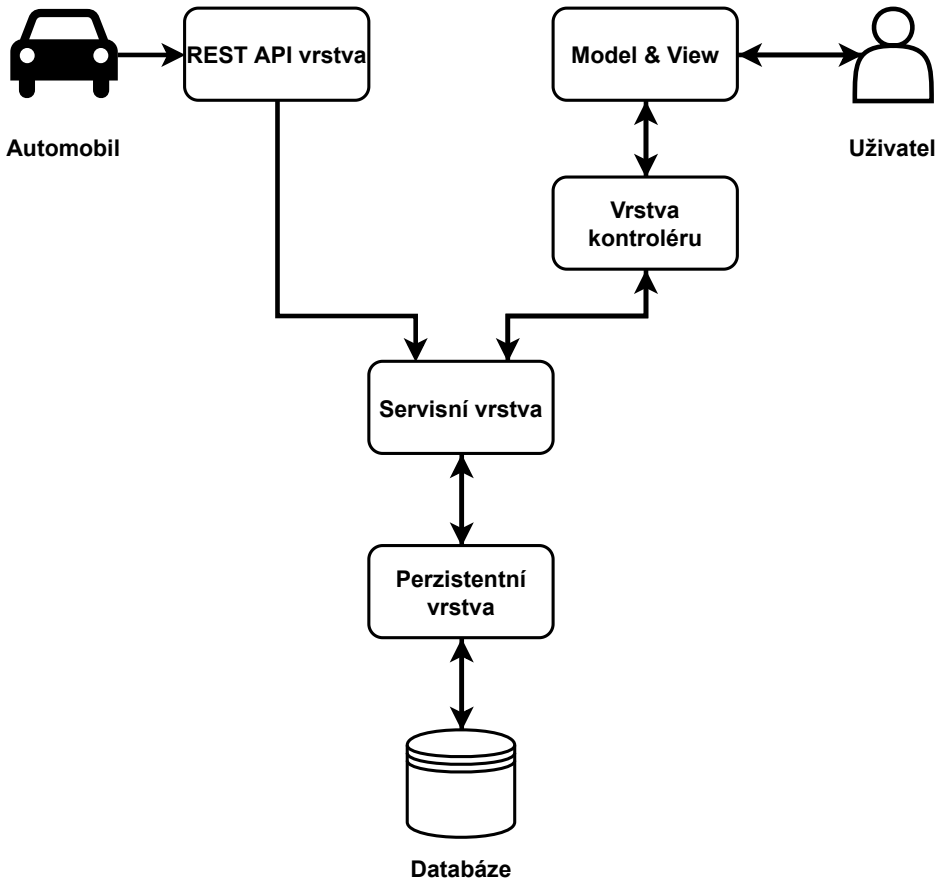
`Controller` zpracuje tento požadavek a vyvolá požadovanou akci, tzn. zobrazí danou stránku. Požadovaná data jsou do komponenty `View` dodána komponentou `Model`.

4.2.2 N–vrstvá architektura

N-vrstvá architektura (anglicky N-tier architecture) se dělí na více logických vrstev, podle jejich účelu. Na obrázku 4.3 je schéma celkové architektury tohoto projektu, v němž jsou propojeny architektury MVC a N-vrstvá architektura. Vrstva REST API zpracovává diagnostická data, které pochází z automobilu. Ve vrstvě kontroléru jsou vytvářeny objekty typu DTO (Data Transfer Object). S nimi bude následně v servisní vrstvě provedeno mapování na entitní objekty. Tyto objekty budou předány do perzistentní vrstvy, která komunikuje s databází, do níž budou tyto objekty uloženy. Data mohou samozřejmě proudit i opačným směrem. Tzn. pokud bude chtít uživatel získat data z databáze, data projdou všemi dříve popsanými vrstvami, bude provedeno potřebné mapování objektů a nakonec budou data předána z vrstvy kontroléru pomocí komponenty Model do komponenty View a budou přehledně zobrazena v tomto uživatelském rozhraní.

4.3 Vývoj serverové části aplikace

Pro serverovou část aplikace je zažitý anglický termín Backend. Tato část aplikace zajišťuje komunikaci se serverem, veškerou aplikační logiku a komunikaci s databází. Nemá žádný atraktivní vzhled, ten je později možné dodat uživatelským rozhraním [22]. Jednotlivé vrstvy serverové části aplikace budou popsány v následujících podkapitolách.



Obr. 4.3: N–vrstvá architektura.

4.3.1 REST API vrstva

REST API vrstva je v tomto projektu zastoupena třídou `RestEventsController`. Ta má za úkol zpracovávat data, odesílaná modulem v automobilu, která přichází na definovanou adresu. Data přichází ve formátu JSON JavaScript Object Notation. Jedná se o jednoduchý, pro člověka dobře čitelný formát. Jeho pravděpodobně největší předností je, že není závislý na použití konkrétního programovacího jazyka [23]. Tato data jsou přemapována na DTO a je ověřena jejich validita. Pokud je přijatý VIN kód aplikaci známý a zároveň přijatá data obsahují všechny povinné atributy, je přijaté hlášení přiřazeno k danému vozidlu a uloženo do databáze.

4.3.2 Vrstva kontroléru

Funkcí komponenty `Controller` je zpracovávat požadavky uživatele. Je také zodpovědný za vytváření odpovědí uživateli. Právě tato komponenta je zodpovědná za zobrazení jednotlivých stránek na příslušných adresách. V tomto projektu je kontrolérů hned několik, podle jejich určení:

- `CarsController` – Kontrolér pro registraci automobilu a zobrazení informací o automobilu.
- `LoginController` – Kontrolér pro zobrazení přihlašovacího formuláře.
- `ProfileController` – Kontrolér pro zobrazení a zpracování dat o uživateli.
- `RegistrationController` – Kontrolér pro zobrazení registračního formuláře a zpracování zadaných dat.

V každém z těchto kontrolérů dochází k mapování HTTP metod jako jsou například metody GET nebo POST. To je poměrně snadné s využitím frameworku Spring, který pro dříve uvedené metody nabízí předdefinované anotace `@GetMapping` a `@PostMapping`. Využití těchto anotací ve třídě `CarsController` je na následující ukázce:

```
@Controller
public class CarsController {
    private final PersonService personService;
    private final CarService carService;

    @Autowired
    public CarsController(PersonService personService,
    CarService carService) {
        this.personService = personService;
        this.carService = carService;
    }

    @GetMapping(path = "/add_car")
    public ModelAndView showCar(Model model) {
        CarDto carDto = new CarDto();
        model.addAttribute(carDto);
        return new ModelAndView("addCar");
    }
    @PostMapping(path = "/add_car")
    public ModelAndView addCar(@ModelAttribute("car")
    @Valid CarDto carDto, Principal principal) {
        String email = principal.getName();
        Person person = personService.findByEmail(email);
        carService.saveCar(person, carDto);
        return new ModelAndView("redirect:/garage");
    }
}
```

Anotace `@Controller` na počátku označuje tuto třídu jako komponentu kontrolér. Anotace `@Autowired` zajišťuje vložení závislosti se servisní vrstvou, kterou představují v tomto případě třídy `PersonService` a `CarService`. `@GetMapping` slouží pro vyřízení HTTP požadavků typu `GET`. Metoda `showCar` přiřazená k této anotaci má za úkol zobrazit HTML šablonu `addCar` na adrese serveru, která končí `/add_car`. Anotace `@PostMapping` slouží pro práci s HTTP požadavky typu `POST`. V tomto případě je k této anotaci přiřazena metoda `addCar`, která nejdříve zkontroluje správnost vstupních dat pomocí anotace `@Valid` a následně zavolá metodu `saveCar` ze servisní vrstvy, která provede potřebná mapování a po zavolání perzistentní vrstvy uloží informace o automobilu.

4.3.3 Servisní vrstva

Úkolem servisní vrstvy je zajistit komunikaci s perzistentní vrstvou a vrstvou kontroléru. Také je zodpovědná za mapování objektů DTO na objekty entitní a naopak, podle toho, zda se informace do databáze ukládají nebo se z ní získávají. Třídy, které plní funkce servisní vrstvy se ve frameworku Spring označují anotací `@Service`. V tomto projektu jsou těmito komponentami třídy:

- `AddressService` – Obsahuje metody pro práci s adresami uživatelů.
- `CarEventsService` – Obsahuje metody pro práci s hlášeními, které zasílá automobil.
- `CarService` – Obsahuje metody pro práci s informacemi o automobilu.
- `MailService` – Obsahuje metody nutné pro sestavování a odesílání emailů.
- `MyUserDetailsService` – Obsahuje metody potřebné k autentizaci uživatele při přihlášení.
- `PersonService` – Obsahuje metody pro práci s daty o uživateli.

4.3.4 Perzistentní vrstva

Perzistentní vrstva je zodpovědná za poskytování dat z databáze a ukládání dat do databáze. Komunikuje se servisní vrstvou a databází. V tomto projektu se perzistentní vrstva skládá ze tříd, definujících entitní objekty a rozhraní, která implementují rozhraní `JpaRepository`. Toto rozhraní umožňuje zapisovat, upravovat a získávat data z databáze. V této aplikaci byla pro práci s databází vytvořena následující rozhraní komunikující s jednotlivými tabulkami v databázi:

- `AddressRepository`,
- `CarEventsRepository`,
- `CarRepository`,
- `ConfirmationTokenRepository`,
- `PersonRepository`,

- `PrivilegesRepository`,
- `RolesRepository`.

Jako entitu zde označujeme objekt, který slouží jako předloha pro tabulku v databázi. Jednotlivé atributy objektu jsou namapovány jako sloupce tabulky. Pro každou entitu byla vytvořena nová třída s odpovídajícím názvem a atributy. Vznikly tak třídy:

- `Person`,
- `Car`,
- `ConfirmationToken`,
- `CarEvents`,
- `Address`,
- `Privilege`,
- `Role`.

Na následující ukázce kódu je vytvoření entity `Car`, která bude uložena do tabulky `cars`.

```
1  @Entity
2  @Table(name = "cars")
3  public class Car implements Serializable {
4
5      @Id
6      @GeneratedValue(strategy = GenerationType.IDENTITY)
7      @Column(name = "id_car", nullable = false)
8      private Long idCar;
9      @Column(name = "car_identifier", nullable = false)
10     private String carIdentifier;
11     @ManyToOne(fetch = FetchType.LAZY)
12     @JoinColumn(name = "id_person")
13     private Person person;
14     @OneToMany(fetch = FetchType.LAZY, mappedBy = "car")
15     private List<CarEvents> carEvents = new ArrayList<>();
```

Tato technika se nazývá ORM (Object–Relational Mapping) a umožňuje práci s databází bez nutnosti psát všechny SQL příkazy. Anotace použité v tomto úryvku kódu jsou definovány ve frameworku Hibernate. `@Entity` označuje tuto třídu jako entitu. `@Table` dovoluje specifikovat detaily, které má tabulka mít, v tomto případě je zde specifikován pouze název `cars`. `@Id` označuje, který atribut má být primárním klíčem tabulky, v tomto případě je jím `id_car`. `@GeneratedValue` zaručuje, že hodnota atributu, nad nímž je použit, se bude generovat automaticky. Zde je jako strategie generování zvoleno `IDENTITY`. `@Column` se používá pro specifikování

detailů daného sloupce. V ukázce je jím určen název sloupce a parametr `nullable = false` říká, že daný parametr musí nabývat nějaké hodnoty, nemůže být ponechán prázdný. `@ManyToOne` vytváří vztah N:1 mezi entitami `Car` a `Person`, přičemž `@JoinColumn` definuje jako cizí klíč `id_person`. To znamená, že jedna entita `Person` může mít přiřazeno více entit `Car` a přiřazení proběhne podle atributu `id_person`. `@OneToMany` vytváří vztah 1:N mezi entitami `Car` a `carEvents`. Následně pak bude možné k jedné entitě `Car` přiřadit více entit `carEvents`.

Další třídy s entitami mají podobnou strukturu, liší se svými atributy a vztahy s dalšími entitami, jsou v nich ale použity obdobné anotace frameworku Hibernate. Dále všechny obsahují bezparametrický konstruktor, konstruktor se všemi parametry, gettery a settery pro všechny atributy.

4.3.5 Implementace zabezpečení

S rostoucím počtem webových služeb vzrůstá také počet útoků na tyto služby. Proto je potřeba implementovat do těchto služeb zabezpečovací mechanismy, které odpovídají aktuálním doporučením pro danou technologii. V tomto projektu nebyl řešen tzv. hosting. Služba tak běží pouze na lokálním serveru počítače. Proto nebylo potřeba řešit použití protokolu TLS, který slouží pro ověřování důvěryhodnosti webových stránek na základě certifikátů. Co ale bylo potřeba řešit, je autentizace, autorizace a ukládání hesel v zašifrované podobě.

Autentizace je proces, při němž je ověřována identita subjektu. Toto ověření může probíhat např. na základě znalosti hesla, číselného kódu, biometrických vlastností osoby (např. pomocí otisků prstů či skenu duhovky) nebo pomocí hardwaru, jako je například flash disk s potřebnými daty [24]. Autentizace je v tomto projektu řešena pomocí hesla, které si uživatel zvolí při registraci. Toto heslo musí být alespoň 8 znaků dlouhé, musí obsahovat alespoň jedno malé písmeno, alespoň jedno velké písmeno a alespoň jedno číslo. Je také vhodné nevolit jako součást hesla celá slova či osobní informace, jako jsou datum narození, telefonní číslo a podobně [25].

Po zadání hesla do registračního formuláře je heslo nejdříve zhashováno. Úkolem hashovací funkce je vytvořit z jednoho řetězce znaků řetězec jiný, podle předem daného algoritmu. Tato funkce musí být jednosměrná, tedy z výsledného hashe nesmí jít pomocí této funkce získat původní řetězec znaků. Musí být také bezkolizní, to znamená, že ze dvou různých vstupních řetězců nesmí po použití hashovací funkce vzniknout stejný hash [26]. Mezi nejznámější hashovací algoritmy patří například SHA, která má v současnosti 3 verze, nebo MD5. V tomto projektu byl pro hashování hesel použit hashovací algoritmus BCrypt, jenž patří mezi doporučené hashovací algoritmy pro použití ve Spring Boot aplikaci. Ten při hashování používá tzv. sůl. Kryptografická sůl je náhodný řetězec znaků, který je přidán k hashovanému ře-

tězci znaků [27]. Použití soli znemožňuje útok pomocí tzv. duhových tabulek. Po zhashování uživatelského hesla, je heslo uloženo do databáze. Při každém přihlášení je pak zadané heslo opět zhashováno a porovnáno se záznamem daného uživatele v databázi. V případě shody dojde k autentizaci a úspěšnému přihlášení.

Autorizace je proces který zpravidla následuje autentizaci. Při autorizaci je kontrolováno, jakou roli má uživatel, popřípadě jaká práva má v dané aplikaci či systému konkrétní uživatel [24]. Pro tyto účely byly v databázi vytvořeny tabulky `Roles` a `Privileges`. V tabulce `Roles` jsou uloženy záznamy o rolích, které existují v systému [28]. Jsou to role `Admin` a `User`. Těmto rolím jsou přiřazena práva z tabulky `Privileges`. V této tabulce existují práva pro čtení a zápis. Administrátorovi s rolí `Admin` jsou přiřazena oba tato práva, běžnému uživateli s rolí `User` je přiřazeno pouze právo pro čtení. Nový uživatel dostane automaticky při registraci roli `User` a tomu odpovídající práva. Pro přiřazení rolí jednotlivým uživatelům vznikla tabulka `person_roles` a pro přiřazení konkrétních práv jednotlivým rolím vznikla tabulka `roles_privileges`.

Všechny výše popsané postupy byly nakonfigurovány ve třídě `SecurityConfig`, kde bylo definováno, jaký algoritmus má být použit pro hashování hesel, které stránky mají být přístupné pro nepřihlášené uživatele (například stránky s registrací a přihlášením), které stránky mají být přístupné pro přihlášené běžné uživatele (například stránka s automobilem, profilem) a které stránky mají zůstat pouze pro administrátora (stránka s databází).

4.4 Vývoj uživatelského rozhraní aplikace

V původní semestrální práci byla pro implementaci uživatelského prostředí použita architektura MVC (Model-View-Controller). V této navazující práci měl být použit framework `React.js`. Od tohoto bylo ale po dodatečné konzultaci s vedoucím práce upuštěno. Vyvíjení backendové a frontendové části projektu zvlášť je velmi výhodné, pokud na projektu pracuje několik vývojářů naráz. Mohou si práci rozdělit a následně tyto části spojit do jednoho celku například pomocí nástroje Docker [29]. V tomto konkrétním případě jsme ale s vedoucím práce usoudili, že bude lepší pojmout vše jako jeden projekt a uživatelské rozhraní implementovat jako architekturu MVC s použitím frameworku Thymeleaf. Hlavními důvody pro tuto změnu jsou především jednodušší implementace funkcionalit, předchozí zkušenosti s technologií Thymeleaf a programovacími jazyky HTML a CSS. Pro jednotlivé stránky byly tedy vytvořeny HTML šablony, k nim dále odpovídající soubory CSS pro stylizaci. Pro tvorbu horního menu aplikace byl jako základ použit volně dostupný soubor CSS z webu Free Frontend, který poskytuje základní prvky pro tvorbu uživatelského rozhraní zdarma [30].

5 Výsledný vzhled a fungování aplikace

Pro lepší představu o fungování a vzhledu aplikace byly vytvořeny snímky obrazovky pro všechny významné akce v aplikaci. V následující kapitole budou tyto snímky uvedeny a blíže popsány.

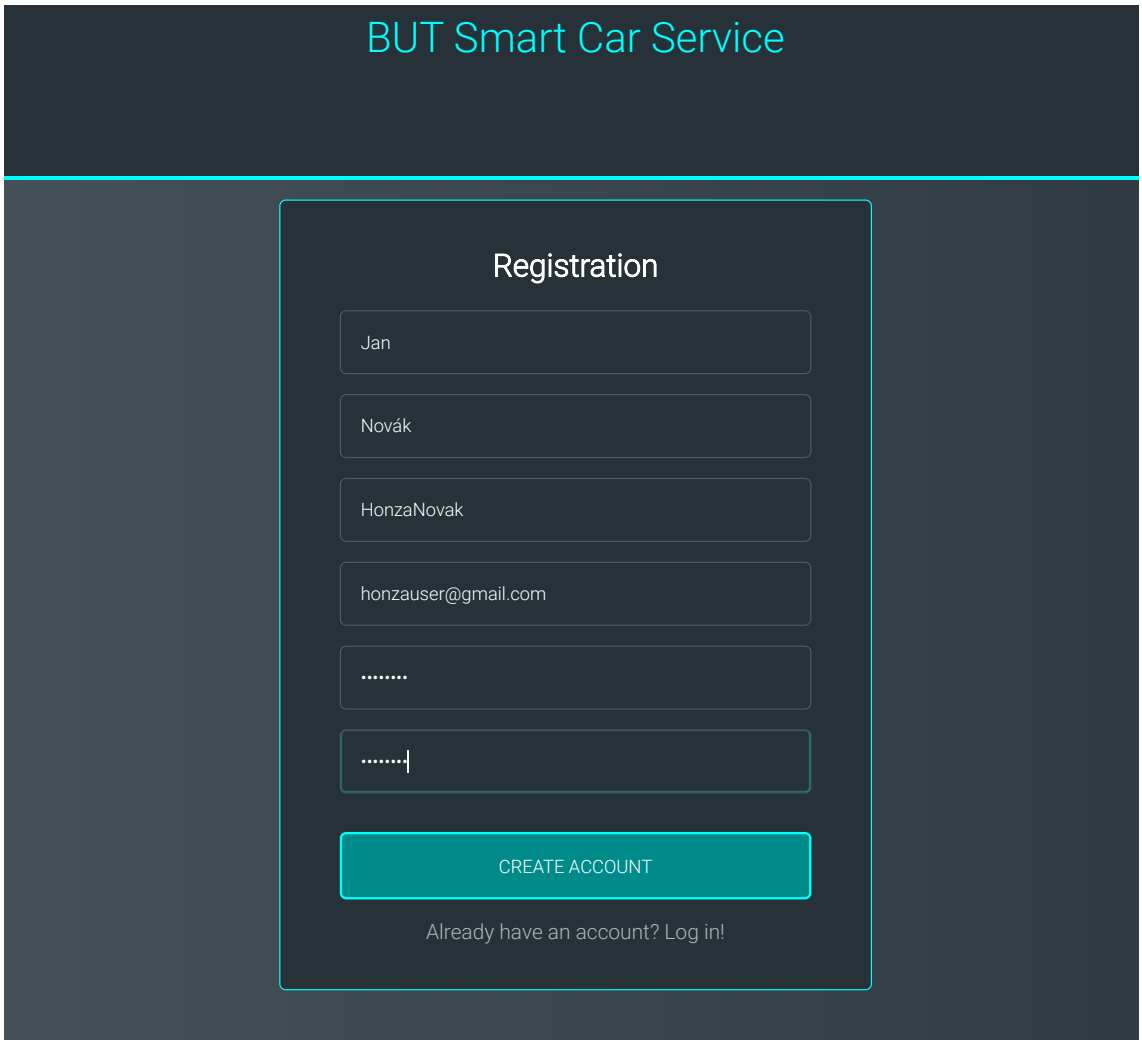
5.1 Lokalizace aplikace

Přestože je práce vypracována v češtině, pro aplikaci byla zvolena anglická lokalizace. Hlavním důvodem bylo, že aplikace měla původně být součástí většího celku, ve kterém by byla anglická lokalizace použita globálně. Ve Spring Boot aplikaci je však možné jednotlivé lokalizace jednoduše přidávat. V případě potřeby by tak bylo možné přidat českou lokalizaci. Lze tak učinit pomocí frameworku Thymeleaf a několika komponent přidaných do aplikace. Hlavně je však potřeba přidat soubor s požadovanými překlady pro jednotlivé zprávy [31].

5.2 Registrace a přihlášení uživatele

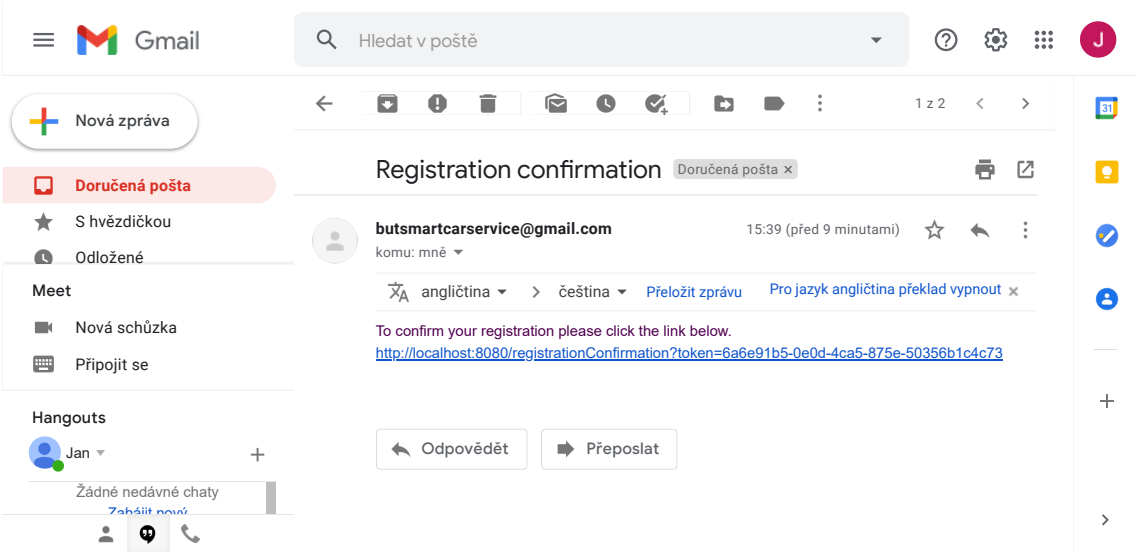
Aby uživatel mohl tuto aplikaci používat, je nutné, aby se nejprve zaregistroval. K tomu slouží registrační formulář na obrázku 5.1. Pro demonstraci byla vytvořena nová e-mailová adresa u společnosti Google a spolu s ostatními smyšlenými informacemi byla vyplněna do registračního formuláře. Uživatel musí vyplnit všechna požadovaná pole, což je kontrolováno jak v HTML atributem `required`, tak při vytváření objektů v serverové části, aplikace pomocí anotací `@NotNull` a `@NotEmpty`. Pro ověření, zda si odpovídají obě zadaná hesla, byla vytvořena vlastní anotace `@PasswordMatches`. Po kliknutí na tlačítko `CREATE ACCOUNT` je požadavek odeslán a je ověřeno, zda je zadaný e-mail unikátní. V případě, že se již v databázi nachází, je o tomto faktu uživatel informován chybovým hlášením a je vyzván k přihlášení namísto registrace. Pokud je e-mail unikátní, jsou zadané informace uloženy do databáze a uživateli je tak vytvořen účet. Tento účet však ještě není aktivní a uživatel se nemůže přihlásit. Uživatel je následně vyzván, aby zkontroloval svoji e-mailovou schránku. Na tu mu byl po úspěšné registraci odeslán e-mail s odkazem jako na obrázku 5.2. Tento odkaz obsahuje potvrzovací token. Po kliknutí na odkaz je uživatel přesměrován na adresu, která vyhodnotí validitu tokenu. V případě, že je daný token správný a jeho platnost nevypršela, uživatelský účet je aktivován a uživatel se nyní může přihlásit.

Pro přihlášení je nutné zadat e-mail a heslo, které si uživatel zvolil při registraci. Přihlašovací formulář je na obrázku 5.3. Pokud uživatel ještě nemá vytvořený účet,

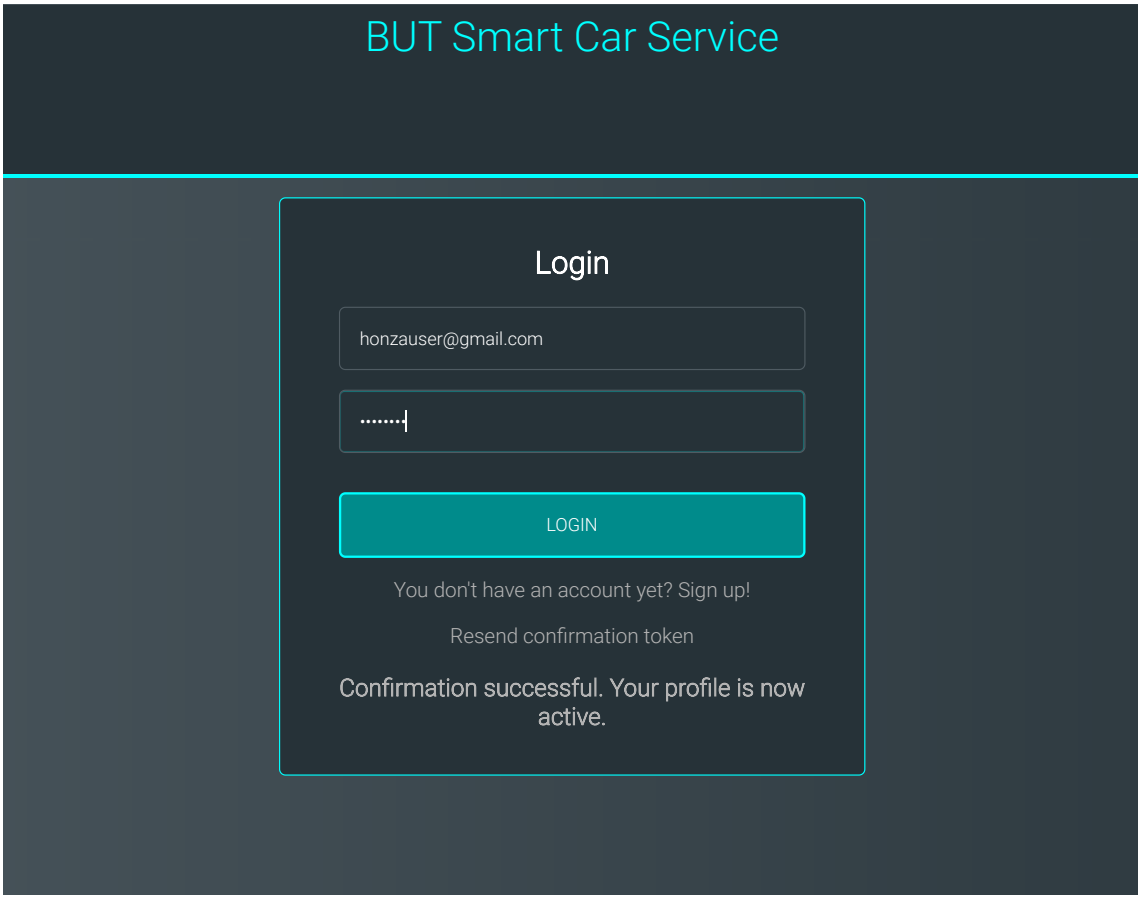


Obr. 5.1: Vyplněný registrační formulář.

je možné se přes odkaz v dolní části formuláře jednoduše dostat na stránku s registračním formulářem a uživatelský účet si vytvořit. Na obrázku 5.3 je ilustrován případ, kdy ověření tokenu proběhlo bez problému. Pokud by ověření neproběhlo v pořádku, je možné nechat si vygenerovat nový token kliknutím na odkaz „Resend confirmation token“, který je v přihlašovacím boxu. Poté je nutné znovu zadat e-mailovou adresu, na kterou má být odkaz s tokenem doručen. Uživatel je opět vyzván ke kontrole své e-mailové schránky, do které mu bude doručen hypertextový odkaz s nově vygenerovaným tokenem. Následně je uživatel schopen proces ověření zopakovat.



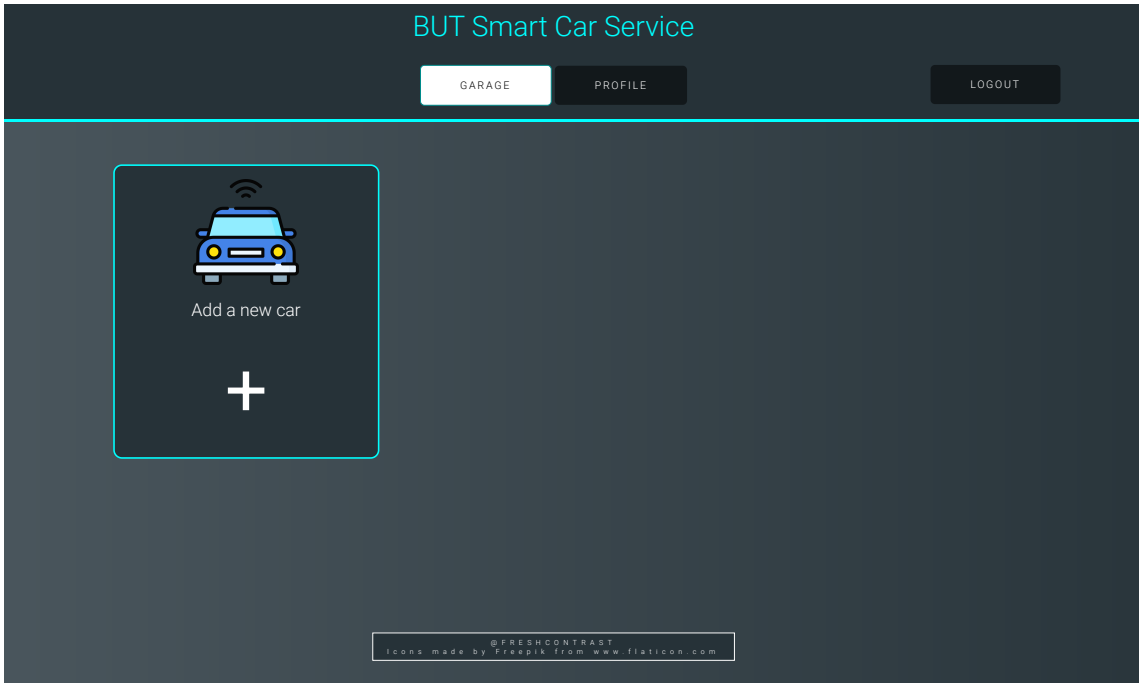
Obr. 5.2: Potvrzovací e-mail.



Obr. 5.3: Přihlašovací formulář.

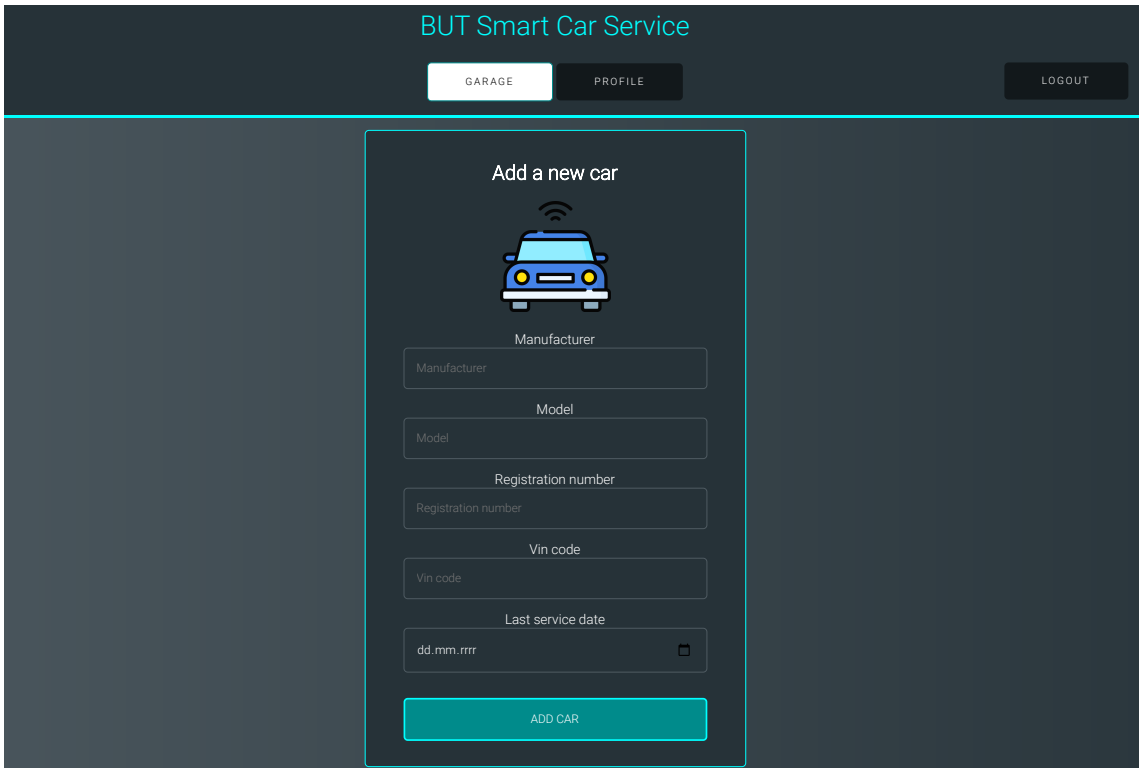
5.3 Registrace automobilu

Po úspěšném přihlášení je uživatel ihned přesměrován na stránku zobrazující jeho automobily. Na rozdíl od předchozích stránek, které jsou přístupné i bez přihlášení, je zde již k dispozici jednoduché menu v horní části stránky. Tlačítko „Garage“ vede na již zmiňovanou stránku, která reprezentuje virtuální garáž s automobily uživatele a je zobrazena na obrázku 5.4. Tlačítko „Profile“ vede na stránku s informacemi o uživateli. V tomto případě je účet nový, nemá tedy zaregistrována žádná vozidla. Registraci lze jednoduše provést kliknutím na pole „Add a new car“. Následně je



Obr. 5.4: „Garáž“ uživatele bez automobilů.

zobrazena stránka s formulářem pro registraci nového vozidla. Ikonka automobilu je převzata z webu flaticon [32], který poskytuje ikony a další již vytvořené grafiky volně k použití. Automobil je poté možné jednoduše přidat. Je nutné vyplnit výrobce automobilu, model, registrační číslo automobilu a VIN kód, který je jedinečný pro každý vyrobený automobil. Dále je také nutné vyplnit datum poslední návštěvy servisu, aby služba mohla poskytovat informace o servisním intervalu. Pro kontrolu validity VIN kódu byla vytvořena anotace `@VinCodeConstraint`. Formulář pro přidání automobilu je na obrázku 5.5. Po vyplnění všech požadovaných položek, je nový automobil uložen do databáze a v uživatelském rozhraní je přidán do „garáže“ uživatele. Počet automobilů, které může jeden uživatel zaregistrovat, není omezen.



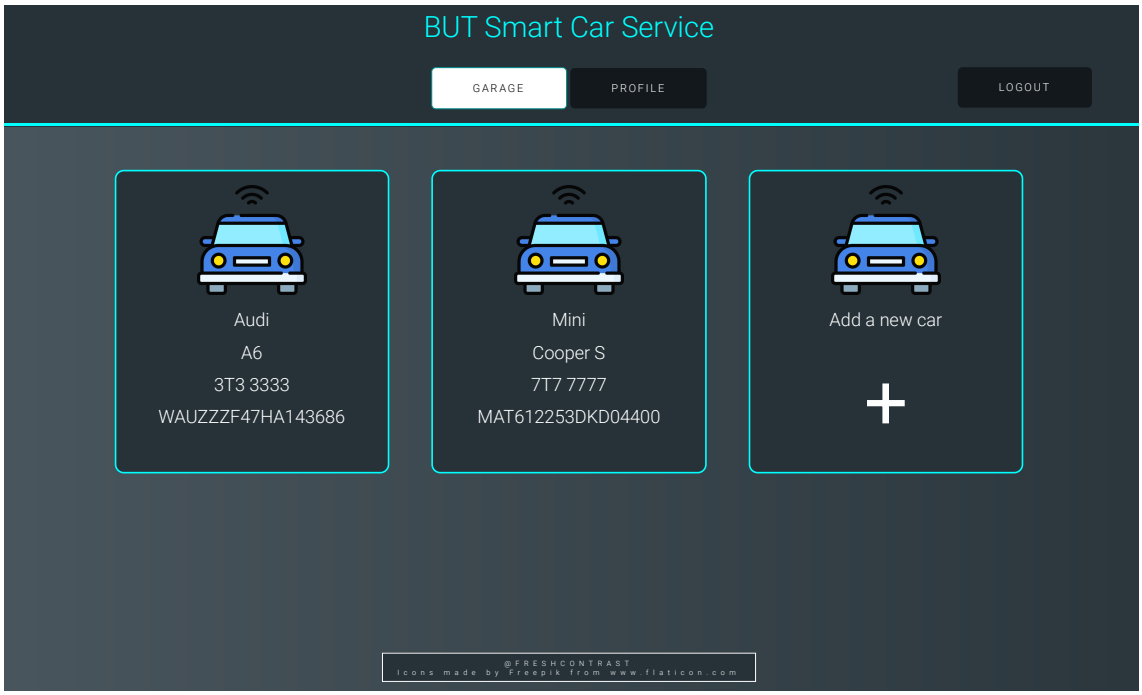
Obr. 5.5: Registrace automobilu.

Po registraci se již automobily zobrazují v „garáži“ uživatele. Na obrázku 5.6 jsou vyobrazeny dva zaregistrované automobily s kompletně smyšlenými daty.

5.4 Zpracování dat z automobilu

Diagnostická a informativní data o automobilu jsou uživateli dostupná po kliknutí na vybraný automobil v uživatelově „garáži“. V následujícím případě byl pro demonstraci vybrán fiktivní automobil Audi A6 z obrázku 5.6. Cílem tohoto projektu nebylo vytvořit řešení zařízení v automobilu, které by odesílalo data směrem na server, proto byl pro posílání požadavků na server v tomto projektu použit nástroj Postman. Postman je velmi užitečná aplikace pro tvorbu API (Application Programming Interface). Snadno a rychle odesílá HTTP požadavky, umožňuje automatizované testování a poskytuje data o vytvářeném API, aby bylo možné vylepšovat jeho výkon a architekturu [33]. Všechna data jsou posílána ve formátu JSON na danou adresu REST API a následně zpracovávána.

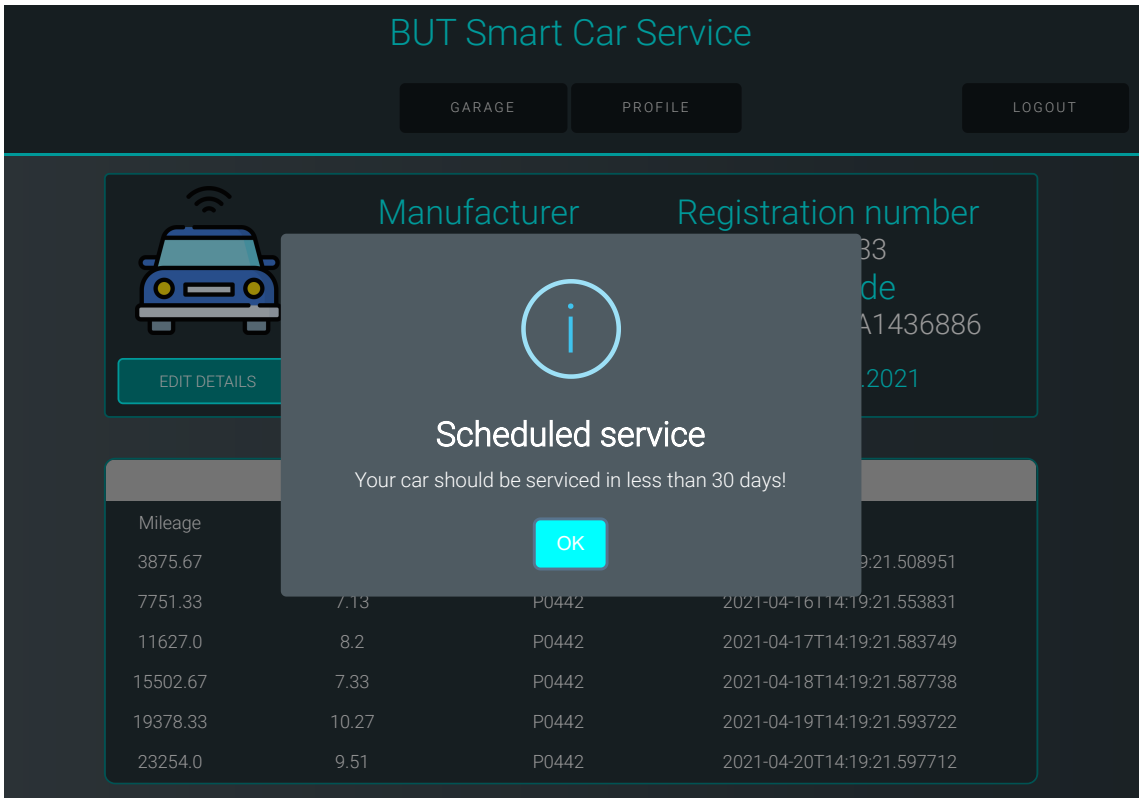
Po kliknutí na vybraný automobil se uživateli zobrazí detaily o automobilu. V tomto případě se jako první zobrazí varování. Protože při registraci automobilu bylo vyplněno poslední datum návštěvy servisu datum 25.4.2020, aplikace upozorní



Obr. 5.6: „Garáž“ uživatele s registrovanými automobily.

uživatele, že za dobu kratší než 30 dní by měl navštívit servis kvůli pravidelné kontrole. Interval mezi návštěvami servisu je nastaven na jeden rok a aplikace kontroluje jednou denně všechny zaregistrované automobily v databázi. V případě, že se servisní interval některého z automobilů blíží ke svému konci, je jeho majitel upozorněn nejdříve e-mailem a poté i v aplikaci, když kontroluje svůj automobil. Po kliknutí na tlačítko `OK` sice varování zmizí, pokud ale uživatel nevyplní datum nové návštěvy servisu, bude upozorněn pokaždé, když navštíví stránku s detaily o svém automobilu. Upozornění bylo vytvořeno pomocí nástroje SweetAlert2. SweetAlert2 je nástroj vytvořený v jazyce JavaScript, který lze použít pro upozornění na webových stránkách, místo klasické funkce `alert`, kterou obsahuje programovací jazyk JavaScript. Je volně dostupný ze stránek vývojářů, či repozitáře vývojářů na platformě GitHub a je možné ho poměrně dobře uzpůsobovat a stylizovat [34]. Upozornění na blížící se konec servisního intervalu je na obrázku 5.7.

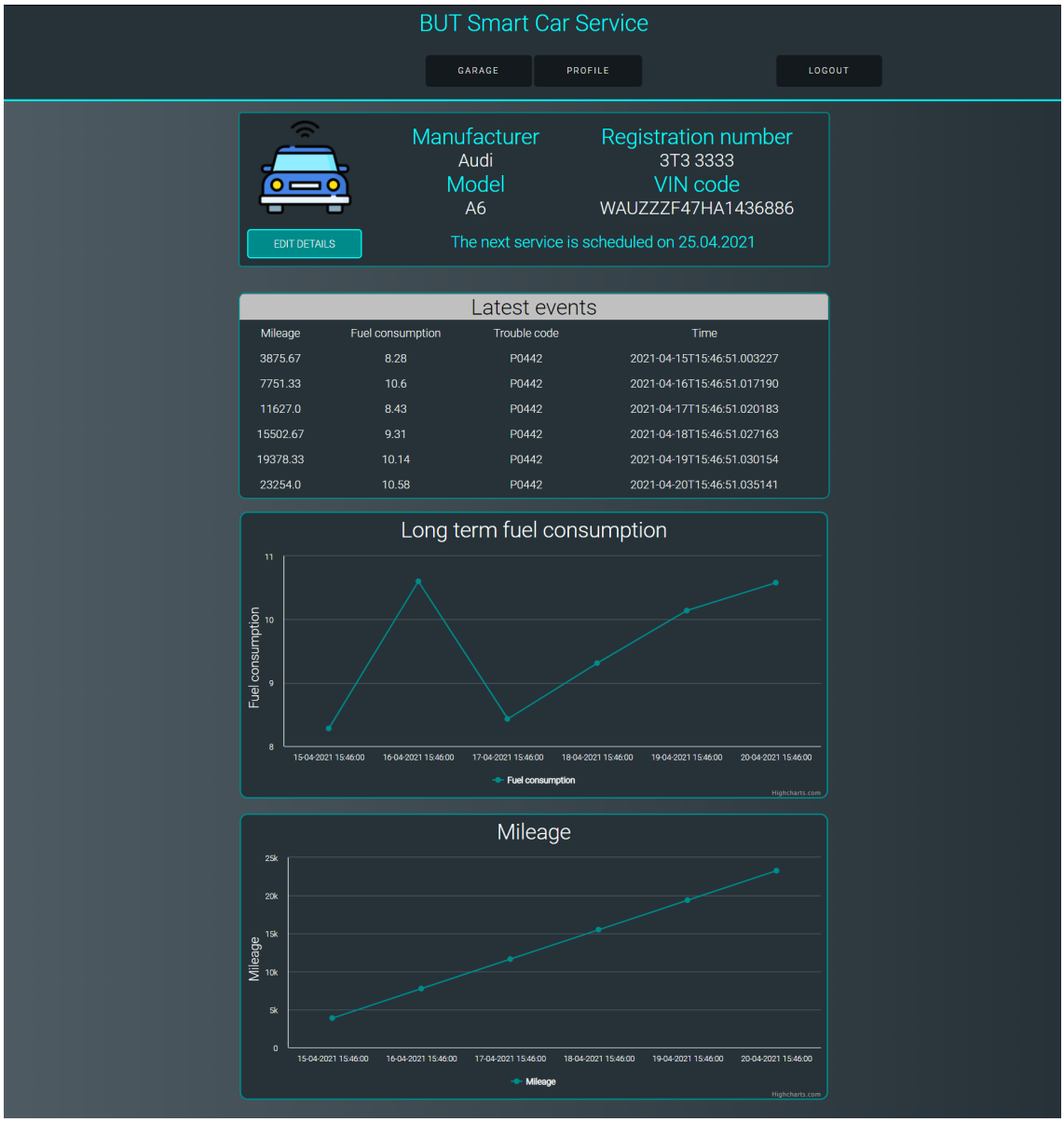
Celá stránka s detaily o automobilu je na obrázku 5.8. V horní části stránky s detaily o automobilu jsou obecné informace, které uživatel zadal do formuláře při registraci automobilu. Tyto informace je možné upravit po kliknutí na tlačítko `Edit details`. Kromě uživatelem zadaných informací je zde také hlášení o konci servisního intervalu. Níže je přehledná tabulka, která obsahuje detaily o jednotlivých hlášeních, která automobil zaslal. V prvním sloupci je uveden nájezd automobilu, ve druhém průměrná spotřeba za daný časový úsek, ve třetím chybový kód DTC.



Obr. 5.7: Upozornění na konec servisního intervalu.

V tomto případě byl mezi námi vygenerovaná data přidán chybový kód P0442, který je poměrně běžný a znamená, že byl detekován malý únik v systému odpařování emisí (EVAP). V posledním sloupci je pak čas, kdy bylo hlášení doručeno serveru.

Níže je možné nalézt grafy s dlouhodobou spotřebou a nájezdem automobilu. Data uvedená v těchto grafech byla náhodně vygenerována. Po najetí na jednotlivé body grafu kurzorem myši se zobrazí detaily o daném bodě. Uživatel má tak přehledně zobrazeno, jakou spotřebu měl v uplynulých dnech a kolik má jeho automobil najeto kilometrů. K realizaci těchto grafů byl vybrán nástroj Highcharts. Highcharts je open source nástroj umožňující realizaci nejrůznějších grafů, map a tabulek na webových stránkách. V tomto projektu byl implementován jako skript v jazyce JavaScript. Výsledné grafy jsou poté poměrně jednoduché na implementaci, je možné je dobře stylizovat a upravovat, díky propracovanému API, které Highcharts poskytuje [35].

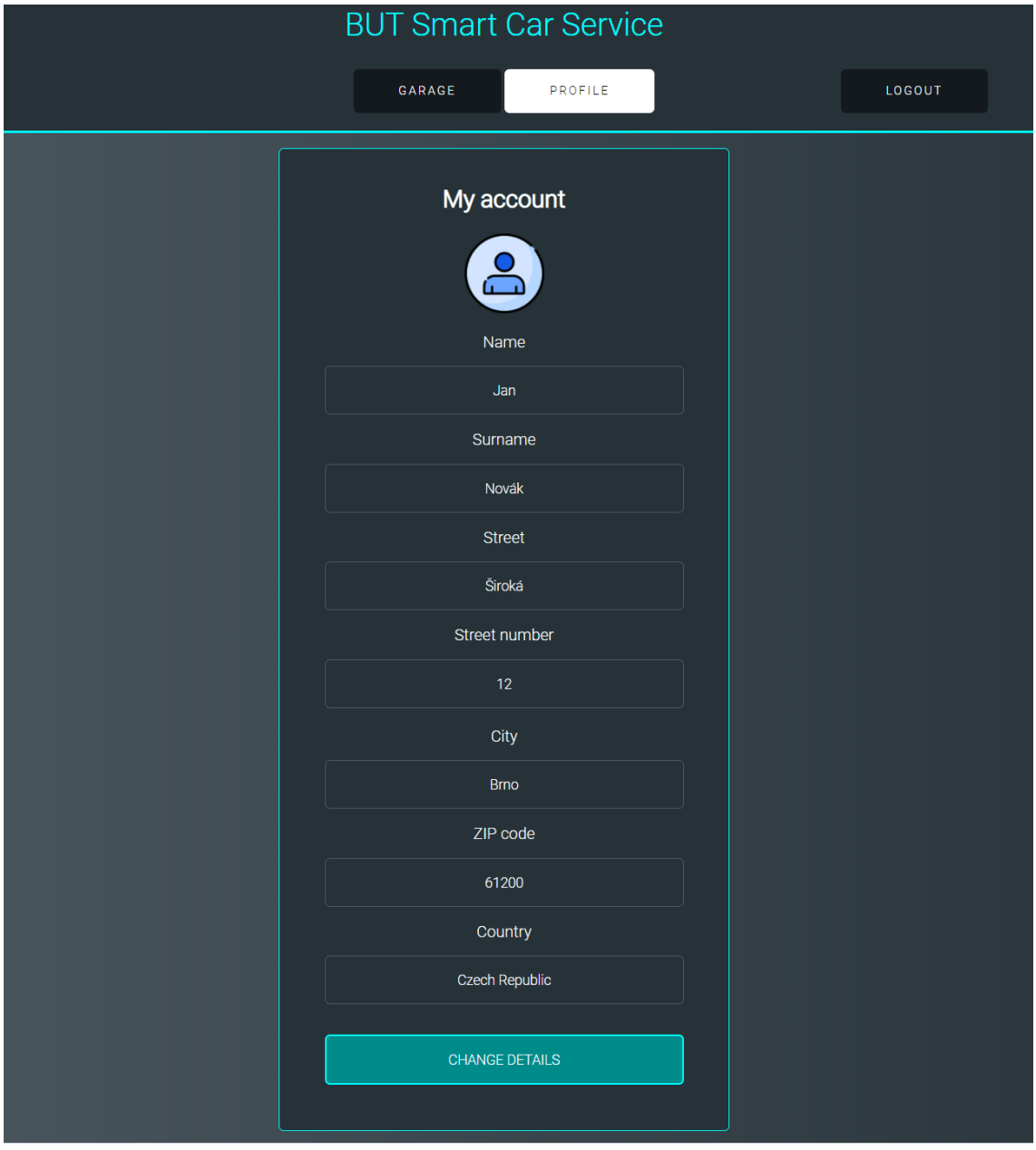


Obr. 5.8: Stránka s detaily o automobilu.

5.5 Informace o uživateli

V horním menu se kromě tlačítek `Garage` a `Logout` nachází také tlačítko `Profile`. Po kliknutí na toto tlačítko je uživatel přesměrován na stránku, kde je možné vyplnit adresní údaje. Tyto údaje je možné využít například pro pozdější fyzickou korespondenci. Při prvním zadávání adresy, kdy ještě není adresa uživatele v databázi, je uživateli zobrazen formulář, do kterého vyplní své údaje. Položky `Name` a `Surname` jsou předvyplněny z dat, která uživatel zadal pří registraci. Po odeslání formuláře jsou data uložena do databáze a uživateli je zobrazen jeho profil s nově

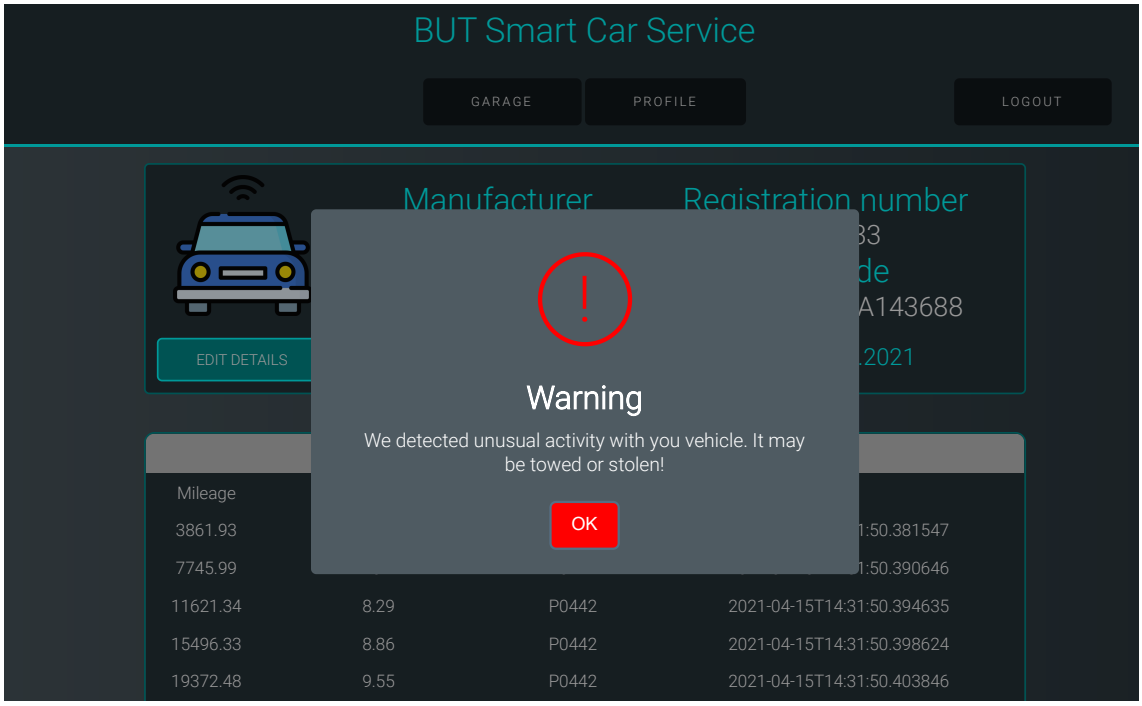
zadanými informacemi. Na této konkrétní stránce není možné do jednotlivých polí cokoliv psát. Všechna pole formuláře je však možné změnit po kliknutí na tlačítko `CHANGE DETAILS`. Poté jsou jednotlivá pole otevřena pro úpravu a uživatel může své údaje podle libosti změnit. Ikona uživatele v horní části formuláře byla, stejně jako ikona automobilu použitá v projektu, převzata z webu flaticon, který poskytuje zdarma ikony pro nekomerční použití. Stránka s profilem uživatelem „HonzaNovak“, který byl vytvořen pro demonstrační účely je na obrázku 5.9.



Obr. 5.9: Stránka s profilem uživatele.

5.6 Upozornění na neoprávněnou manipulaci s vozidlem

Mezi požadavky na službu bylo mimo jiné uvedeno, že kromě poskytování diagnostických a informativních dat má služba také upozorňovat uživatele v případě neoprávněné manipulace s jeho automobilem. Služba tedy vyhodnocuje příchozí hlášení z automobilu a v případě, že mezi přijatými daty najde upozornění na neoprávněnou manipulaci, ihned informuje uživatele e-mailem, že je něco v nepořádku a měl by svůj automobil zkontrolovat. Dále je také v aplikaci zobrazeno upozornění, které taktéž upozorňuje uživatele na podezřelou aktivitu s jeho automobilem. Upozornění na neoprávněnou manipulaci s automobilem bylo, podobně jako upozornění na blížící se konec servisního intervalu, implementováno pomocí nástroje SweetAlert2. Upozornění na podezřelou aktivitu automobilu v aplikaci je zobrazeno na obrázku 5.10.



Obr. 5.10: Upozornění na neoprávněnou manipulaci s vozidlem.

6 Závěr

V rámci této bakalářské práce byla vytvořena webová aplikace, jejíž serverová část byla vyvinuta jako Spring Boot aplikace, která je propojena s databází. Uživatelské rozhraní bylo vytvořeno jako HTML šablony s kaskádovými styly, do nichž byly v případě potřeby implementovány skripty v jazyce JavaScript. Spojení serverové části s uživatelským rozhraním bylo provedeno pomocí architektury MVC, aby byla aplikace vyvíjena jako jeden celistvý projekt.

Aplikace v současné podobě umožňuje registraci uživatele i s potvrzením registrace, které je provedeno pomocí tokenu. Tento token je obsažen v hypertextovém odkazu, který je uživateli odeslán e-mailem. Tokeny mají své časové omezení, pokud by uživatel token použil příliš pozdě, je možné si službou nechat vygenerovat nový token a zaslat ho na emailovou adresu uživatele. Po úspěšné registraci a potvrzení je uživatel schopen se do aplikace přihlásit. Přihlášený uživatel je schopen zaregistrovat do služby své automobily, nad nimiž má poté díky uživatelskému rozhraní přehled. Služba umožňuje uživateli sledovat poslední události s vozidlem, které vozidlo zasílá na server. Tyto události obsahují informace o nájezdu a spotřebě automobilu, chybových kódech a času, kdy byla tato data odeslána. V rámci této práce nebylo cílem vyvinout koncové zařízení v automobilu, které je potřebné pro zasílání dat na server. Pro testovací účely byl proto použit nástroj Postman, pomocí kterého byla testovací data zasílána na rozhraní REST API. Informace o spotřebě a nájezdu jsou uživateli k dispozici také v podobě grafů, které byly implementovány pomocí knihovny Highcharts napsané v jazyce JavaScript. V případě, že služba detekuje podezřelou aktivitu s vozidlem, je uživatel upozorněn jak emailem, tak výstrahou přímo v aplikaci. Uživatel je také upozorněn v případě, že se blíží konec servisního intervalu a on by se svým vozem měl navštívit autoservis. Uživatel také může vyplnit své korespondenční údaje, které by byly využity v případě, že by internetová korespondence emailem nebyla dostačující a bylo by nutné použít fyzické korespondence.

V současné době je jako databázový systém použit H2. To je poměrně jednoduchý systém, který může běžet jen v paměti zařízení, není tedy umístěn na serveru. Byl použit z důvodu jednodušší implementace a přehlednějšího zobrazení databáze pomocí „H2 Console“. Pro použití v produkčních aplikacích je nevhodný, avšak pro projekty tohoto typu, které nemíří do produkčního použití a jsou jen jakýmisi prototypy, je poměrně výhodný právě z důvodu jednoduché implementace.

Oproti semestrální práci bylo také pokročeno v oblasti bezpečnosti. Byly přidány nástroje pro autentizaci a autorizaci. Hesla jsou nyní hashována pomocí funkce BCrypt a v této podobě ukládána do databáze. Bez přihlášení je možné přistupovat pouze na stránky, které jsou k tomu určeny (např. stránka s přihlášením, registrací).

Jsou rozlišeny dva typy uživatelů. Administrátor a běžný uživatel. Běžný uživatel má omezená práva, administrátor má práva vyšší a je například schopen manipulovat s databází.

Literatura

[1] KOČÍ, Petr. *Diagnostika a testování automobilů: učební text : studijní materiály pro studijní program Mechatronika*. Ostrava: Vysoká škola báňská - Technická univerzita Ostrava, 2012. ISBN 978-80-248-2609-7.

[2] CULMER, Kris. From the archive: How Volkswagen pioneered computer check-ups. *Autocar* [online]. 2 June 2020 [cit. 2020-12-01]. Dostupné z: `https://www.autocar.co.uk/car-news/features/archive-how-volkswagen-pioneered-computer-check-ups`.

[3] *OBDII: PAST, PRESENT & FUTURE* [online]. 2011 [cit. 2020-10-21]. Dostupné z: `http://www.obdii.com/articles/OBDII_Past_Present_Future.html`.

[4] KLOC, Pavel. *Diagnostika EOBD* [online]. [cit. 2020-10-28]. Dostupné z: `https://www.puvodni-blog.autodiagnostik.cz/autodiagnostika-eobd/`.

[5] BLECHA, Jiří. *Automobilová diagnostika - zapojení diagnostické zásuvky* [online]. 2014 [cit. 2020-10-21]. Dostupné z: `http://www.jb-elektronik.cz/diagnostika-zapojeni_diagnosticke_zasuvky.php`.

[6] HORPATZKÁ, Michaela. *S*ystémy sériové diagnostiky motorových vozidel [online]. Brno, 2016 [cit. 2020-12-11]. Dostupné z: `https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=128274`. Bakalářská práce. Vysoké učení technické v Brně. Vedoucí práce Ing. Martin Beran.

[7] SMITH, Craig. *The Car Hacker's Handbook: A Guide for the Penetration Tester*. San Francisco: William Polock, 2016. ISBN 9781457198847.

[8] ŠŤASTNÝ, Petr. *Diagnostika automobilu na mobilních telefonech s OS Google Android* [online]. Brno, 2013 [cit. 2020-10-26]. Dostupné z: `https://dspace.vutbr.cz/xmlui/bitstream/handle/11012/26578/final-thesis.pdf?sequence=11&isAllowed=y`. Bakalářská práce. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací. Vedoucí práce Ondřej Lutera.

[9] SLAVSKAYA, Polina. Vehicle bus ISO/OSI model visualized. *Medium* [online]. Oct 12, 2017 [cit. 2021-5-13]. Dostupné z: `https://medium.com/@thrashmetalsoul/vehicle-bus-iso-osi-model-visialized-d47460bc158f`.

[10] *International Journal of Scientific & Engineering Research: A Survey on Automotive Diagnostics Protocols* [online]. 8. Karnataka, India, 2017 [cit. 2020-12-01]. ISSN 2229-5518. Dostupné z: https://www.ijser.org/researchpaper/A-Survey-on-Automotive-Diagnostics-Protocols.pdf.

[11] MIKULIČ, Martin. *P*rogram pro sběr dat z vozidla s využitím diagnostického rozhraní [online]. Praha, 2015 [cit. 2020-12-11]. Dostupné z: https://dspace.cvut.cz/bitstream/handle/10467/63729/F2-DP-2015-Mikulic-Martin-Martin%20Mikulic_Diplomova%20praca.pdf?sequence=-1. Diplomová práce. České vysoké učení technické v Praze. Vedoucí práce Ing. Vojtěch Klír Ph.D.

[12] WHITE, Stephen A. *Business Process Modeling Notation* [online]. [cit. 2020-11-14]. Dostupné z: https://is.muni.cz/el/1433/jaro2014/PV165/um/46771256/pr_06_bpmn.pdf.

[13] OTTE, Lukáš. *Databázové systémy: Databázové (datové) modely a modelování* [online]. Ostrava, 2013 [cit. 2020-11-14]. Dostupné z: http://projekty.fs.vsb.cz/463/edubase/VY_01_044/Databázové%20systémy/02%20Text%20pro%20e-learning/Databázové%20systémy%2003.%20Databázové%20(datové)%20modely%20a%20modelování.pdf.

[14] KHUNARA, Rohit. *Programming with Java*. 1. Delhi: Vikas Publishing House PVT, 2014. ISBN 978-93259-7839-3.

[15] DE OLIVEIRA, Claudio Eduardo, Greg L. TURNQUIST a Alex ANTONOV. *Developing Java Applications with Spring and Spring Boot: Practical Spring and Spring Boot solutions for building effective applications*. Birmingham: Packt Publishing, 2018. ISBN 9781789539134.

[16] RICHARDSON, Leonard a Sam RUBY. *RESTful Web Services*. Sebastopol: O'Reilly Media, 2007. ISBN 9780596554606.

[17] OTTINGER, Joseph B., Jeff LINWOOD a Dave MINTER. *Beginning Hibernate: For Hibernate 5*. 4th. New York: Apress, 2016. ISBN 9781484223192.

[18] KOSEK,Jiří *HTML: tvorba dokonalých WWW stránek : podrobný průvodce. Praha: Grada, 1998. Průvodce (Grada). ISBN 80-716-9608-0. Dostupné z: http://htmlguru.cz/*

[19] FRAIN, Ben. *Responsive Web Design with HTML5 and CSS: Develop future-proof responsive websites using the latest HTML5 and CSS techniques*. 3rd. Birmingham, UK: Packt Publishing, 2020. ISBN 9781839219795.

[20] Difference between IntelliJ Idea and Eclipse. *JavaTpoint* [online]. Noida, India [cit. 2020-12-04]. Dostupné z: https://www.javatpoint.com/intellij-vs-eclipse.

[21] DECK, Paul. *Spring MVC: A Tutorial*. Quebec: Brainy Software, 2013. ISBN 9780980839654.

[22] PASTORINO, Marcelo. *Frontend vs. backend: what's the difference?* Pluralsight [online]. [cit. 2021-03-31]. Dostupné z: https://www.pluralsight.com/blog/software-development/front-end-vs-back-end#:~:text=What%20is%20back%2Dend%20development,these%20apps%20render%20server%2Dside.

[23] *Introducing JSON* [online]. [cit. 2021-03-31]. Dostupné z: https://www.json.org/json-en.html.

[24] VOLENEC, Martin. *Bezpečnost webových aplikací* [online]. Pardubice, 2020 [cit. 2021-5-10]. Dostupné z: https://dk.upce.cz/handle/10195/75522. Diplomová práce. Univerzita Pardubice. Vedoucí práce Roman Diviš.

[25] SONI, Rakesh. Password Security Best Practices in 2020. *Business 2 Community* [online]. February 9, 2020 [cit. 2021-5-10]. Dostupné z: https://www.business2community.com/cybersecurity/password-security-best-practices-in-2020-02282074#:~:text=Generally%2C%20the%20minimum%20password%20length,minimum%20length%20to%2014%20characters.&text=It%20should%20have%20at%20least,uppercase%2C%20numbers%2C%20and%20symbols.

[26] MILLINGTON, Sam. Hashing a Password in Java. *Baeldung* [online]. January 9, 2021 [cit. 2021-5-10]. Dostupné z: https://www.baeldung.com/java-password-hashing.

[27] PARASCHIV, Eugen. Registration with Spring Security: Password Encoding. *Baeldung* [online]. September 3, 2020 [cit. 2021-5-10]. Dostupné z: https://www.baeldung.com/spring-security-registration-password-encoding-bcrypt.

[28] PARASCHIV, Eugen. Spring Security: Roles and Privileges. *Baeldung* [online]. June 28, 2020 [cit. 2021-5-10]. Dostupné z: https://www.baeldung.com/role-and-privilege-for-spring-security-registration.

[29] *Empowering App Development for Developers | Docker* [online]. [cit. 2021-04-04]. Dostupné z: https://www.docker.com/.

[30] BALAA, Karim. *CodePen Home Simple Menu Navigation. Free Frontend* [online]. [cit. 2021-4-4]. Dostupné z: https://codepen.io/karimbalaa/pen/WboBBY/.

[31] Guide to Internationalization in Spring Boot. *Baeldung* [online]. August 31, 2020 [cit. 2021-5-10]. Dostupné z:https://www.baeldung.com/spring-boot-internationalization.

[32] Autonomous Car free icon. In: *Flaticon* [online]. [cit. 2021-04-03]. Dostupné z: https://www.flaticon.com/free-icon/autonomous-car_3253345?term=autonomous%20car&page=1&position=4&page=1&position=4&related_id=3253345&origin=search.

[33] *Postman* [online]. [cit. 2021-04-08]. Dostupné z: https://www.postman.com.

[34] *SweetAlert2* [online]. [cit. 2021-04-14]. Dostupné z: https://sweetalert2.github.io.

[35] *Highcharts* [online]. [cit. 2021-04-14]. Dostupné z: https://www.highcharts.com/blog/products/highcharts/.

Seznam symbolů, veličin a zkratek

API	Application Programming Interface
BPMN	Business Process Model and Notation
CAN	Controller Area Network
CARB	California Air Resources Board
CSS	Cascading Style Sheets
DTC	Diagnostic Trouble Codes
DTO	Data Transfer Object
EEPROM	Electrically Erasable Programmable Read-Only Memory
EGR	Exhaust Gas Recirculation
EOBD	European On-Board Diagnostics
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Enviroment
JPA	Java Persistence API
JSON	JavaScript Object Notation
LIN	Local Interconnect Network Protocol
MIL	Malfunction Indicator Lamp
MVC	Model-View-Controller
MySQL	My Structured Query Language
OBD	On-Board Diagnostic
ORM	Object–Relational Mapping
PID	Parameter IDs
POM	Project Object Model
PWM	Pulse Width Modulation

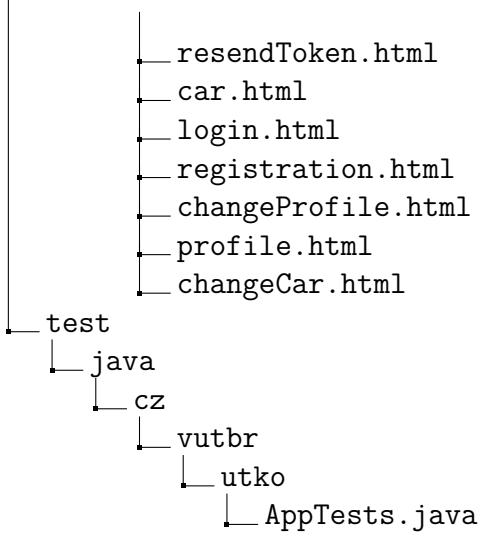
RAM	Random Access Memory
REST	Representational State Transfer
UDS	Unified Diagnostic Services
VIN	Vehicle Identification Number
VPW	Variable Pulse Width
WWW	World Wide Web
XML	Extensible Markup Language

A Zdrojové kódy

Obsah přiloženého souboru `but-smart-car-service.zip`. Je možné ho spustit pomocí vývojového prostředí IntelliJ. Pro funkčnost odesílání e-mailů je nutné do souboru `application.properties` zadat do řádků `spring.mail.username` a `spring.mail.passwor` validní e-mailovou adresu s doménou `gmail.com` a odpovídající heslo. Po spuštění poběží aplikace na adrese lokálního serveru a portu 8080. Program byl otestován na IntelliJ Ultimate verze 2020.2.2.

```
but-smart-car-service
└── src
    └── main
        └── java
            └── cz
                └── vutbr
                    └── utko
                        ├── App.java
                        ├── api
                        │   ├── AddressDto.java
                        │   ├── UpdateCarDto.java
                        │   ├── PersonDetailedViewDto.java
                        │   ├── PersonBasicViewDto.java
                        │   ├── CarEventsDto.java
                        │   ├── CarDto.java
                        │   ├── GetPersonCarDto.java
                        │   ├── ResendTokenPersonDto.java
                        │   └── PersonRegistrationDto.java
                        ├── facade
                        │   └── PersonFacade.java
                        ├── model
                        │   ├── ConfirmationToken.java
                        │   ├── Privilege.java
                        │   ├── Car.java
                        │   ├── Address.java
                        │   ├── Role.java
                        │   ├── Person.java
                        │   └── CarEvents.java
                        ├── exceptions
                        │   ├── UserAlreadyExistsException.java
                        │   └── ResourceNotFoundException.java
                        ├── config
                        │   └── SecurityConfig.java
                        ├── exceptionhandler
                        │   └── RestExceptionHandler.java
                        └── service
                            ├── MyUserDetailsService.java
                            └── MailService.java
```

```
        CarService.java
        CarEventsService.java
        AddressService.java
        PersonService.java
    repositories
        CarRepository.java
        ConfirmationTokenRepository.java
        RolesRepository.java
        PrivilegesRepository.java
        AddressRepository.java
        PersonRepository.java
        CarEventsRepository.java
    controller
        LoginController.java
        PersonRestController.java
        CarEventsRestController.java
        CarsController.java
        ProfileController.java
        RegistrationController.java
    util
        GenericResponse.java
    setup
        SetupData.java
    validation
        PasswordMatchesValidator.java
        PasswordMatches.java
        VinCodeConstraint.java
        VinCodeValidator.java
    mappers
        PersonMapper.java
resources
    application.properties
    messages.properties
    static
        css
            forms.css
            car.css
            profile.css
            addCar.css
            garage.css
        images
            autonomous-car.png
            user.png
    templates
        garage.html
        addCar.html
```


resendToken.html
car.html
login.html
registration.html
changeProfile.html
profile.html
changeCar.html
test
java
cz
vutbr
utko
AppTests.java